Special Report: The Cambridge Algorithms Workshop

#213 APRIL 1994

1275

# Dr. Dobb's JOURNAL

SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

DDJ's ENCRYPTION CONTEST!

See Page 40

## UNLOCKING ENCRYPTION ALGORITHMS

- BLOCK ENCRYPTION
- LUCAS FUNCTIONS
- SECURE HASH ALGORITHM

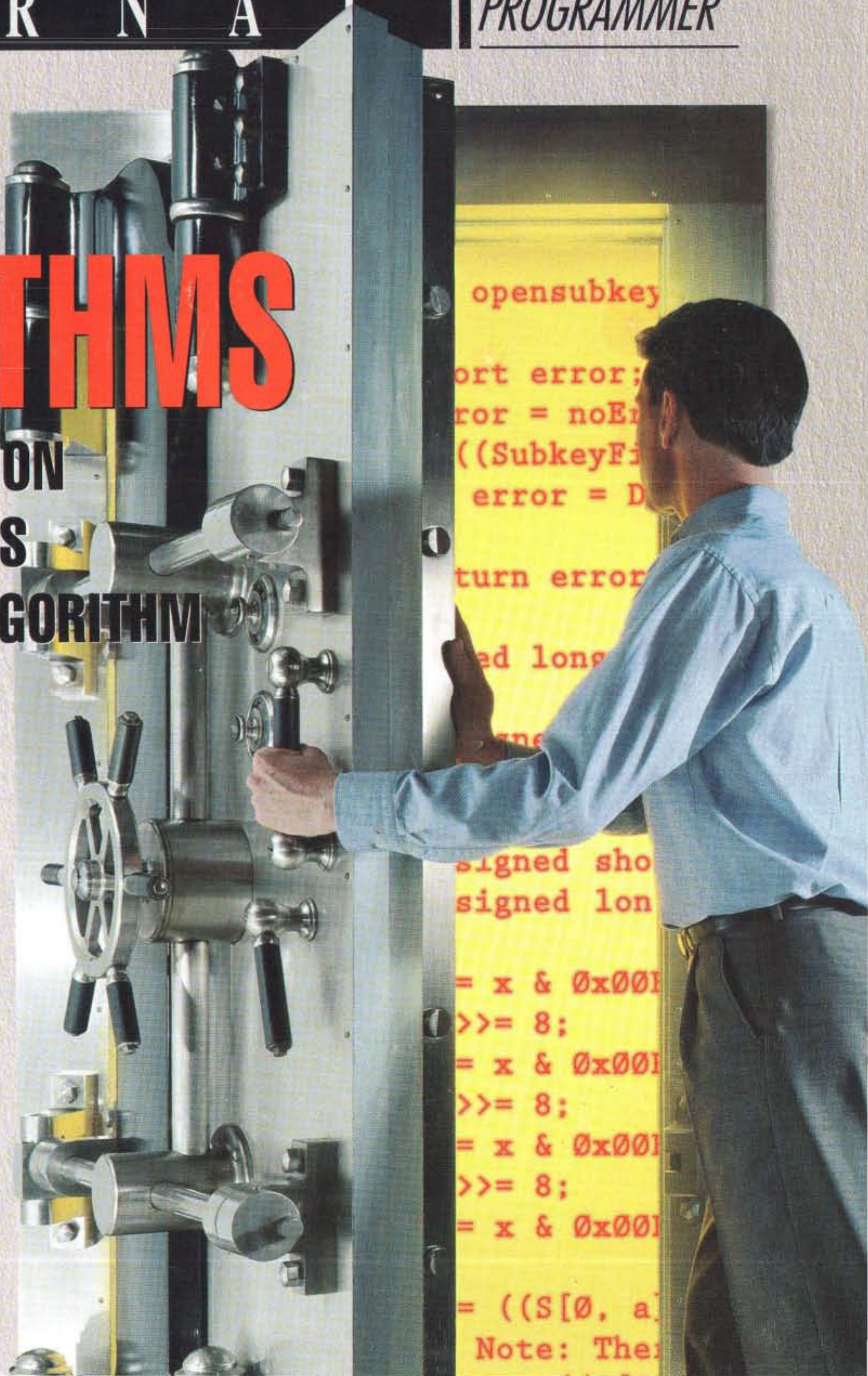
## WAVELET PACKET TRANSFORMS

## DATA ACQUISITION

## EXAMINING WINDOWS HELP AUTHORING TOOLS

## ECHONETS

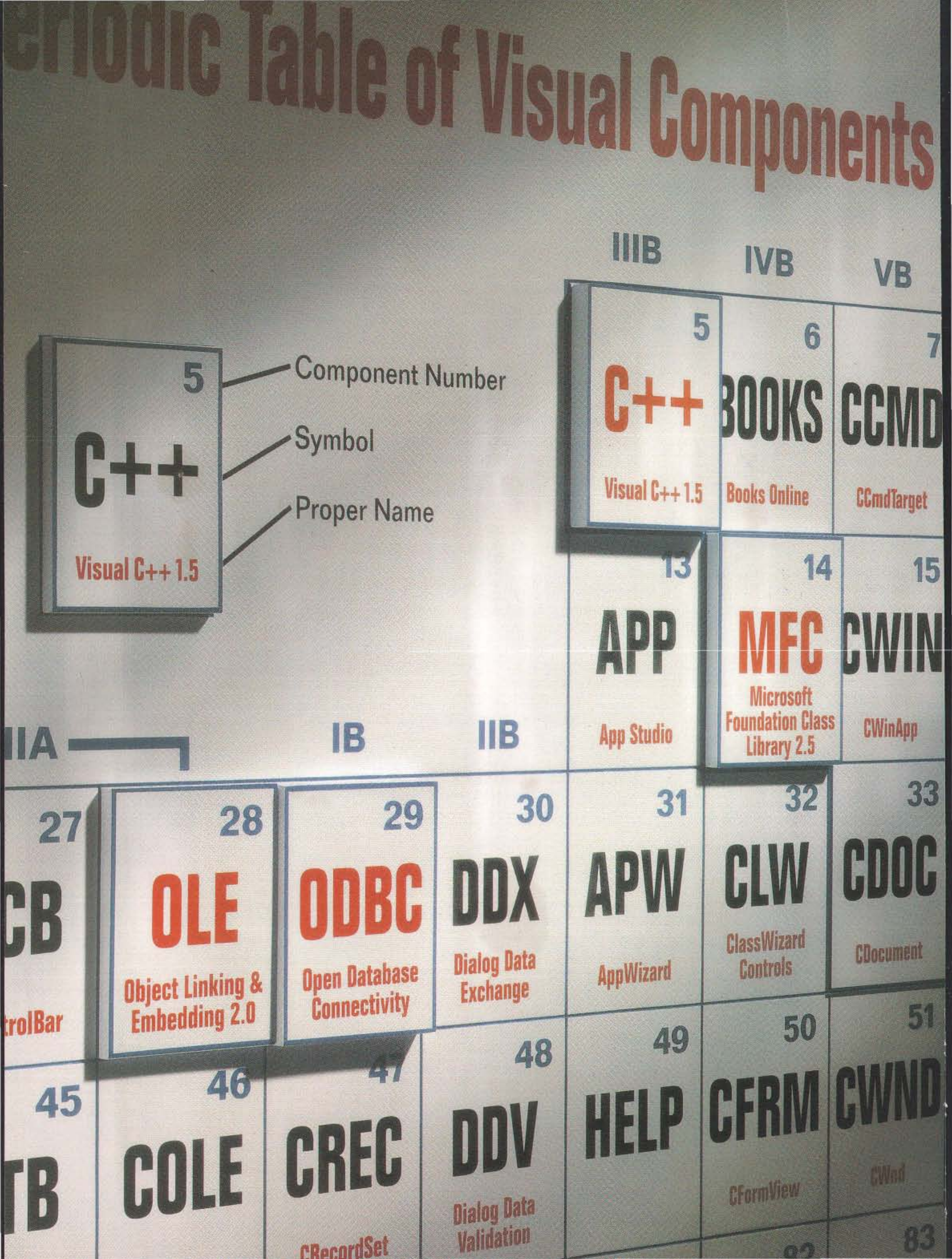
*and more!*



$3.95 ($4.95 CANADA)

04

0 38351 16562 8



riodic Table of Visual Components
5
C++
Visual C++ 1.5
Component Number
Symbol
Proper Name
IIIB
IVB
VB
5
C++
Visual C++ 1.5
6
BOOKS
Books Online
7
CCMD
CCmdTarget
13
APP
App Studio
14
MFC
Microsoft Foundation Class Library 2.5
15
CWIN
CWinApp
IIA
IB
IIB
27
CB
trolBar
28
OLE
Object Linking & Embedding 2.0
29
ODBC
Open Database Connectivity
30
DDX
Dialog Data Exchange
31
APW
AppWizard
32
CLW
ClassWizard Controls
33
CDOC
CDocument
45
TB
46
COLE
47
CREC
CRecordSet
48
DDV
Dialog Data Validation
49
HELP
50
CFRM
CFormView
51
CWND
CWnd
83

# THE BUILDING BLOCKS OF VISUAL DEVELOPMENT.

The elements of next generation applications are found in the components of the Microsoft® Visual C++™ development system, version 1.5 today.

Innovative *wizard* technology allows you to seamlessly meld components into sophisticated applications, in weeks not months. Just remember this simple equation: MFC 2.5, OLE 2.0, ODBC.

That's a perfect formula for building and reusing components. At the nucleus is MFC 2.5 (Microsoft Foundation Class Libraries), the standard API for Windows.™ The new OLE (Object Linking and Embedding) classes make it easier than ever to implement OLE's unprecedented integration power, new drag-and-drop features, visual editing and OLE automation. In all, it provides more than 19,000 lines of code you want, but won't have to write or rewrite.

In addition, the new ODBC (Open Database Connectivity) classes provide access to major databases and data binding without coding. It's a powerful new platform for creating high-performance database applications.

Best of all, your 16-bit MFC applications are portable to the Windows NT™ operating system and future versions of Windows. Now that's good chemistry.

If you want the latest tools for the latest Windows technology, ask for Microsoft Visual C++ 1.5.

Call us now at (800) 434-3980 and we'll make you a catalyst for change today and into the future of Windows operating systems.


**New 16-bit version 1.5 joins the Visual C++ family of development systems for Windows and Windows NT.**

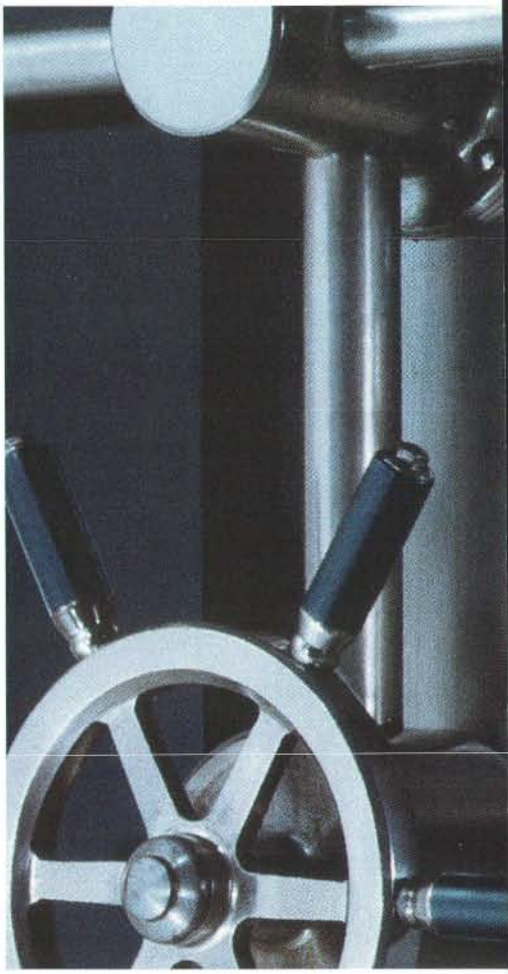
**Microsoft®**

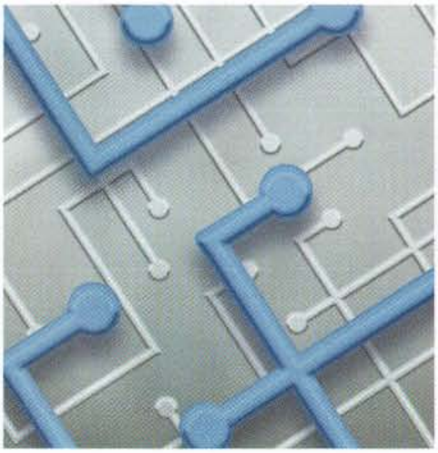
Dr. Dobb's JOURNAL
SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

# CONT

## FEATURES







## EMBEDDED SYSTEMS

APRIL 1994
VOLUME 20, ISSUE 4

# ENTS



## SOURCE CODE AVAILABILITY

As a service to our readers, all source code is available on a single disk and online. To order the disk, send $14.95 (California residents add sales tax) to *Dr. Dobb's Journal*, 411 Borel Ave., San Mateo, CA 94402, call 1-415-358-9500, x240, or use your credit card to order by fax, 1-415-358-9749. Specify issue number and disk format. Code is also available through the DDJ Forum on CompuServe (type GO DDJ), via anonymous FTP from site ftp.mv.com (192.80.84.1) in the /pub/ddj directory, and through DDJ Online, a free service accessible via direct dial at 1-415-358-8857 (1200/2400/9600 baud, 8-N-1).

## NEXT MONTH

In May, we look under the hood of operating systems and microkernels including DOS, OS/2, Windows, UNIX, NetWare, and more.

# Dr. Dobb's JOURNAL

SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

VICE PRESIDENT-PUBLISHER *Peter Hutchinson*

## EDITORIAL

EDITOR-IN-CHIEF *Jonathan Erickson*
MANAGING EDITOR *Tami Zemel*
EXECUTIVE EDITOR *Michael Floyd*
SENIOR TECHNICAL EDITOR *Ray Valdés*
SENIOR PRODUCTION EDITOR *Monica E. Berg*
ASSOCIATE EDITOR *John Dorsey*
ASST. MANAGING EDITOR *Christine de Chutkowski*
ART DIRECTOR *Michael Hollister*
CONTRIBUTING EDITORS *Al Stevens, Tom Genereaux, Andrew Schulman, Ray Duncan, David Betz, Tom Swan*
EDITOR-AT-LARGE *Michael Swaine*
PRODUCTION MANAGER *Amy Shulman Lesovoy*
COVER PHOTOGRAPHER *Michael Carr*

## CIRCULATION

DIRECTOR OF CIRCULATION *Jerry Okabe*
CIRCULATION MANAGER *Michael Poplardo*

## ADMINISTRATION

ACCOUNTING SUPERVISOR *Renate Kernke*
ACCOUNTS RECEIVABLE *Wendy Ho*

## MARKETING/ADVERTISING

ASSOCIATE PUBLISHER *Cynthia C. Sandor*
ADVERTISING COORDINATOR *Sara D. Wood*
*DDJ* BUSINESS MANAGER *Lisa Tarlton*
MARKETING MANAGERS *Valerie Dow, Mike Kuehl*
ACCOUNT MANAGERS see page 136

## MILLER FREEMAN INC.

CHAIRMAN OF THE BOARD *Graham J.S. Wilson*
PRESIDENT/CEO *Marshall W. Freeman*
EXECUTIVE VP/COO *Thomas L. Kemp*
SENIOR VICE PRESIDENTS *H. Verne Packer, Wini D. Ragus, Donald A. Pazour*
VICE PRESIDENT/SOFTWARE *Regina Starr Ridley*
VICE PRESIDENT/CFO *Warren Ambrose*
VICE PRESIDENT/ADMIN. *Charles H. Benz*
VICE PRESIDENT/PRODUCTION *Andrew A. Mickus*
VICE PRESIDENT/CIRCULATION *Jerry Okabe*

**Miller Freeman, Inc.**
*A publication of M&T Publishing, division of Miller Freeman Inc., member of the United Newspapers Group.*

***DR. DOBB'S JOURNAL*** (USPS 307690) is published monthly by Miller Freeman, Inc., 411 Borel Ave., San Mateo, CA 94402-3522; 415-358-9500. Second-class postage paid at San Mateo and at additional entry points.

**ARTICLE SUBMISSIONS:** Send manuscripts and disk (with article, listings, and letter to the editor) to *DDJ* Submissions, 411 Borel Ave., San Mateo, CA 94022-3522.

***DDJ* ON COMPUSERVE:** Type GO DDJ.

**SUBSCRIPTION:** $29.97 for 1 year; $56.97 for 2 years. Foreign orders must be prepaid in U.S. funds drawn on a U.S. bank. Canada and Mexico: $45.00 per year. All other foreign: $70.00 per year.

**FOR SUBSCRIPTION QUESTIONS,** change of address, and orders, call toll-free **800-456-1215** (U.S. and Canada). For all other countries, call **303-447-9330,** or fax **303-443-5080.** On CompuServe, the customer-service number is 71572,341. Or write, *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80322-6188.

**POSTMASTER:** Send address changes to *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80322-6188. **ISSN 1044-789X. GST (Canada) # R124771239**.

**FOREIGN NEWSSTAND DISTRIBUTOR:** Worldwide Media Service Inc., 30 Montgomery St., Jersey City, NJ 07302; 212-332-7100.




ABP American Business Press
Printed in the USA
The Audit Bureau

# Ten Years After

Until the Tonya Harding/Nancy Kerrigan brouhaha ushered in the notion of tag-team figure skating, it had been at least ten years since anything really interesting happened in ice skating. Even back in the 1984 Winter Olympics, what held the attention of anyone other than hardcore figure-skating devotees was whether or not U.S. figure skater Scott Hamilton's jeans could endure one more Russian split, or if a triple axel would finally dislodge East German Katarina Witt's Jimmy Johnson-like coiffure. In retrospect, it would have been far more interesting had we been able to foresee the current plight of the '84 Winter Olympics host city—Sarajevo, Yugoslavia. And speaking of Russian splits, not only has the past decade brought us the breakup of the U.S.S.R., but the demise of East Germany, Yugoslavia, and other such countries as well.

In the relatively calm world of computer programming, it's hard to believe that ten years have passed since Bjarne Stroustrup introduced the C++ programming language. In the high-octane software-development arena, we've come to believe that things move fast, even though it's taken a decade for C++ to gain little more than a toehold. Stroustrup ostensibly created C++ "to make writing good programs easier and more pleasant for the individual programmer." Still, it's taken the marketing might of companies such as AT&T, Microsoft, Sun, IBM, and Borland to make the language a commercial success. The fact that Smalltalk has been around more than twice as long as C++ with far less success, or that there's so darn much Cobol code still out there, says something about the software-development community's desire to absorb and adopt new languages, not to mention the marketing efforts of C++ vendors.

As it enters its second decade, C++ finally seems entrenched. But even though the language is being used to program everything from PCs to supercomputers, the jury is still out on whether or not Stroustrup's goal of making programmers more productive and software less complex has been—or can be—achieved. The real lesson to be learned here is that software development is a process which does not adapt to change easily. There's almost always too much at stake for programmers to casually pick up one language while discarding another—other than for educational or entertainment reasons. Instead, software development continues to be a process of refinement, with major changes occuring over a decade or more at a time. In short, the real issues don't change—just the marketing hype.

A quick glance at *Dr. Dobb's Journal* ten years ago underscores this. As with this month's issue, the April 1984 *DDJ* focused on cryptography, with C.E. Burton's two-part article entitled "RSA: A Public-Key Cryptography System." Without a doubt, cryptography is more important to a greater number of computer programmers and users now than it was a decade ago. Back then, RSA was still fairly new, and Burton's article was probably the first to bring RSA to the microcomputer platform. Today, RSA is clearly the dominant approach to cryptography. While better techniques may have been developed, it is unlikely that in the near future an alternative will achieve the same degree of commercial success as RSA.

1984 was also the year Judge Harold Greene became a household name, at least in the living rooms and kitchens of AT&T executives and stockholders. It was his consent decree, you'll recall, that led to the breakup of the most powerful telecommunications monopoly in the world. Now, ten years after, competition and innovation is finally beginning to catch fire with the Clinton administration's proposals to eliminate barriers between individual electronic-communication industries. However, proposed mergers between cable operators, Baby Bells, entertainment giants, online services, and the like may lead to electronic networks that dwarf the AT&T of yesteryear.

On the plus side, these mergers will fund the advanced information infrastructures we'll be using in the coming years. On the downside, the prospect looms that monopolistic megacorporations will be as unresponsive to the public well-being as Ma Bell was in the old days. For instance, Southwestern Bell, long the bellwether for the RBOCs when it comes to controversial legislation, is currently pushing for a law that would prevent the Missouri Public Service Commission from challenging phone company profits or rates. A draft of the bill reportedly says that Southwestern Bell "shall not under any circumstances be subject to any complaint or hearing as to the reasonableness of its rates, charges, rentals or earnings." Interestingly, this legislation was proposed on the eve of the telephone company's announcement that it had just achieved its best-ever fourth-quarter earnings. The company, which enjoys a monopoly, claims it needs the money to fund the construction of the information superhighway.

Clearly, both the federal and state governments have a responsibility to protect the public good against voracious proposals like that backed by Southwestern Bell. Likewise, the government needs to guarantee that proposed mergers won't result in a single company controlling both telephone lines and cable throughout an individual region. Competition and innovation have stood us well through the past ten years. They can get us through the next decade, too.

Jonathan Erickson

Jonathan Erickson
editor-in-chief

# Develop Windows Applications Quickly and Easily

## Phase3 Has Everything You Need

### Visual Development

The Phase3 visual development environment provides a comprehensive suite of tools for screen creation. All standard Windows screen objects – push buttons, radio buttons, dialog boxes, etc. – are selectable from icon bars and are dynamically placed and sized on the screen as appropriate for the application. Standard Windows APIs are available from list boxes and are supported with on-line documentation.

### Phase3 Database

Phase3 includes a relationally complete database supplied as a Windows DLL. The database supports complex data relationships and access and data manipulation by any language through a comprehensive suite of supplied database routines. Database integrity is enhanced with rollback recovery and transaction control. The Phase3 data dictionary simplifies data model definition and maintenance.

### Query and Reporting

The Report Writer includes standard features like flexible headers, footers, free text, calculated fields, sort group sections and breaks, and subtotals. Reports can also include bitmaps of drawings and photographic images. Queries are executed with the Database Browser which also allows data entry and manipulation. Railway diagrams assist in query creation, prompting the user for appropriate selections and eliminating syntactical errors.

### Lower CASE, E-R Modeling

Phase3 automates logical data model design with Entity-Relationship modeling. After a user describes entity relationships graphically and enters field descriptions, Phase3 generates the physical database based on an analysis of the entity relationships. Phase3 automatically includes foreign keys in appropriate tables and restructures the database as requirements change. Phase3 suggests appropriate referential integrity constraints to be enforced at runtime.

### Hierarchy Chart

Phase3 maintains a graphical map of an entire application. The Hierarchy Chart includes all windows, dialogs, reports, and code routines. To access the underlying C or Pascal source code, simply point-and-click on any node. Phase3 generated source is easily accessed and extended with user written routines. User code is preserved even if an application is regenerated. The Hierarchy Chart and E-R Diagram provide instant core documentation for an application.

### Help Generator

The Phase3 Help Generator allows the easy creation of a complete Windows application help subsystem including context sensitive on-line help. The Help Generator includes its own text editor that gives complete control over the content, appearance, and branching logic through highlighted trigger text. Phase3 generates a Windows compatible file with an ".HLP" extension that is then accessible from within the Phase3 created application. No other external tools are required.

**"... a thoroughly remarkable product... very impressive"**

Jeff Duntemann
**PC Techniques**

**"I was very impressed with Phase3, and using it made me wish I had learned Windows programming with a tool like this. Now that I know what it has to offer, I'll probably use it to build the frame for most of my programs."**

L. John Ribar
**Windows Tech Journal**

**"Phase3 hides the complexity of Windows programming but still allows an experienced programmer to drill down and extend generated C or Pascal source code, providing ultimate control."**

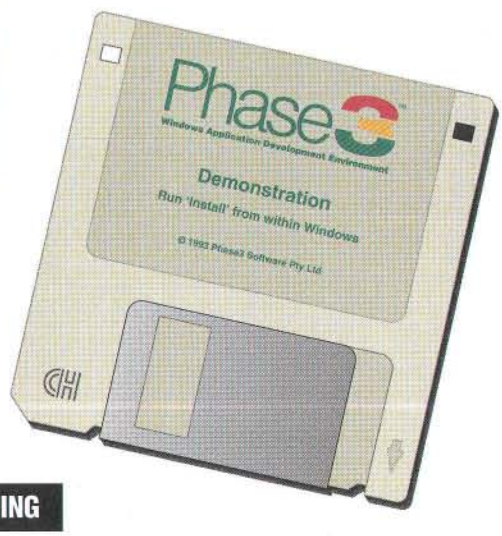
Randy Goodhew
**Computer Software Columnist**

**"Phase3 takes an intuitive, innovative approach to developing Windows applications. It's a pleasure to work with and has greatly boosted our productivity. One of the most important issues for us is that their technical support is excellent."**

Reuben Halevi
**ISoft D&M**

**"It's easy to put together an application with Phase3 – everything's integrated. And the Phase3 database is terrific. It's closer to true relational than anything we've found."**

Michael Erickson
**Prism Business Solutions**




CIRCLE NO. 825 ON READER SERVICE CARD

60 DAY GUARANTEE MONEY-BACK

**Royalty-Free Applications**

## Order Now 800-851-5650

**Or Call for Free Demo Disk**
**Fax (805) 641-9083**

**CALL NOW FOR COMPETITIVE UPGRADE PRICING**

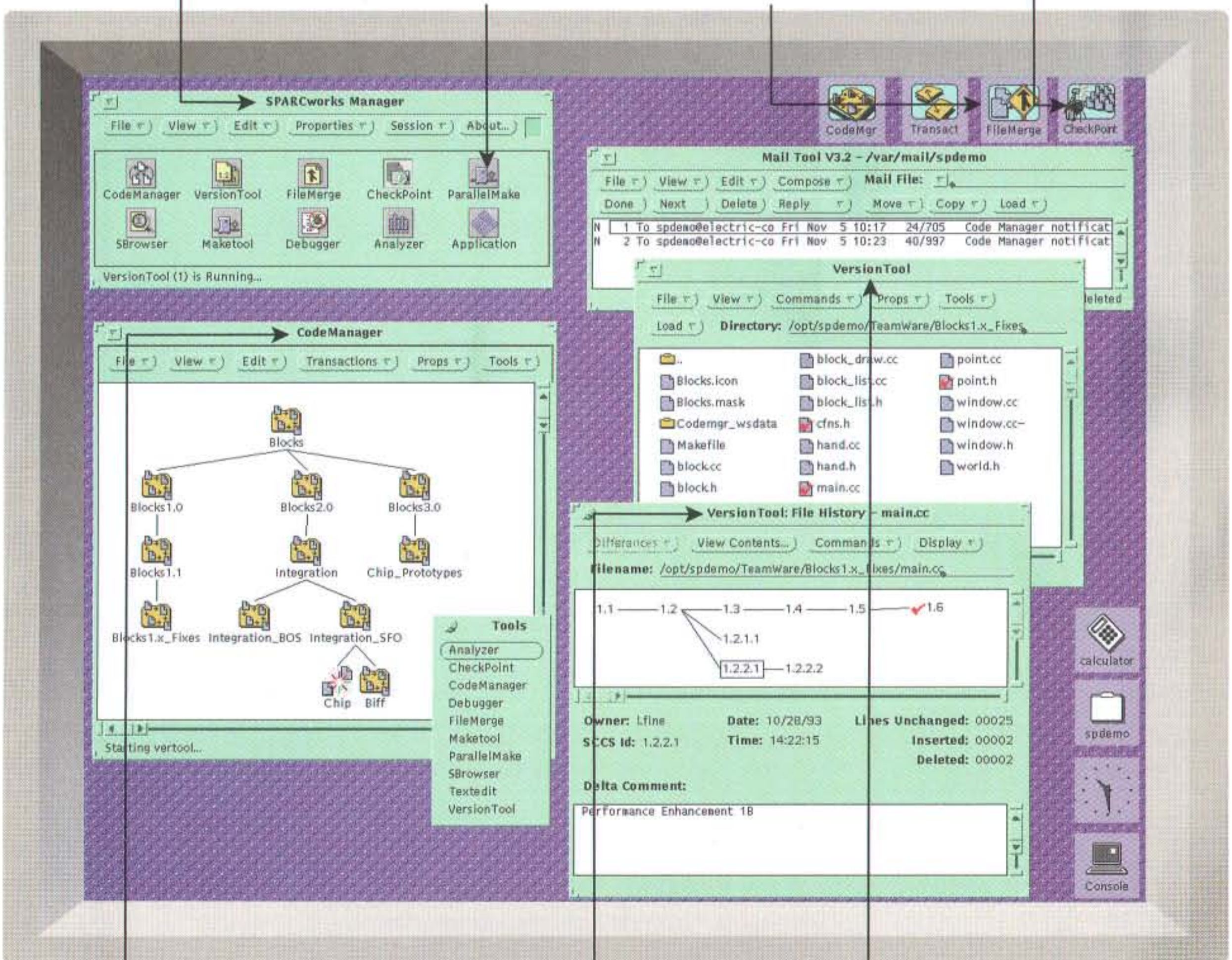
# Can you really look us in the face and say you don't need better code management?

**SPARCworks Manager–**
Coordinate your development sessions and tools. User extensible.

**Parallel Make–**
Dramatically accelerate project builds by leveraging the power of your compute server.

**FileMerge–**
Graphically compare and merge source code. Automatically merge concurrently modified files.

**CheckPoint–**
Capture complete project releases for later retrieval.




**Code Manager–**
Easily organize and integrate work from multiple developers, sites, and platforms. Automatically track and inform of conflicting source code changes.

**Compatible–**
Begin using without any special preparation or administrative requirements.

**Version Tool–**
Accelerate version control. Graphically view source file history as well as concurrent modifications.

Because this is the new face of software development—integrated, organized, productive, and very, very easy to use.

It's called SPARCworks™/TeamWare for the Solaris® operating environment.

And it's from SunPro™, the software development arm of Sun Microsystems, Inc.®

What makes our SPARCworks/TeamWare products so great? Let us answer that question with a few of our own.

How would you like to have multiple developers working on the same source base without getting in each other's way?

Or how about doing source code development, quality assurance and testing, and release engineering at the same time?

Regardless of whether they're in the same country, or even on the same network?

And manage more than one release at a time, with code coming from different locations?

*Without SPARCworks/TeamWare, everybody on your team might as well be locked in their own rooms.*

Perhaps building project components in parallel and integrating them later appeals to you?

We thought so.

You see, SPARCworks/TeamWare can do all that and more. Because it was specifically designed to solve the problems of today's complex, multi-developer, multi-site, multi-platform development projects.

No matter how big or small those projects are.

SPARCworks/TeamWare works with the SPARCworks development environment to be totally scalable and extensible—from small groups to large ones, across geographically distributed sites, and disparate platforms.

## The Features

• Extends SPARCworks development environment • Compatible with C, C++, Pascal, FORTRAN, and Ada • Supports both Solaris 1 and Solaris 2 operating environments • Graphical tracking of multiple releases for easy integration and release management • Graphical inspection and manipulation of multiple source file versions • Captures complete project source file configuration • Compares and merges concurrent source file modifications • Accelerates project builds • Coordinates and manages individual session tasks • ProWorks™/TeamWare available for Solaris development on x86 platforms • Manages code over heterogeneous networks

It's also customizable to each developer's or team's way of working, right down to specific operations.

Which brings us to the next point.

SPARCworks/TeamWare builds on standards your developers already know and use: the UNIX® SCCS utilities, NFS,® and X window system utilities. So there's no ramp-up time, no new system administration, no conversion, and no special databases.

You're ready to run as soon as you load it.

What's more, because SPARCworks/TeamWare uses a graphical approach, developers can work in parallel, and actually see what they—and everybody else—are doing, all at the same time.

And what would you expect to pay for all this?

A lot less than you'd guess.

Under $1,000 a seat.

Call us at 1-800-2SUNPRO for a copy of the SPARCworks/TeamWare Solutions Guide including real life success stories from SPARCworks/TeamWare users, or our 30-day Try and Buy CD which lets you try SPARCworks/TeamWare before you buy it.


So why choose between crude version control tools on one end or overly complex (and expensive) configuration management systems on the other?

Especially now that you've come face to face with something better.

**SunPro**

A Sun Microsystems, Inc. Business



**CIRCLE NO. 903 ON READER SERVICE CARD**

### *Pairing C and C++*

Dear *DDJ*,

In his article "Programming Language Guessing Games" (*DDJ*, October 1993), P.J. Plauger expresses confusion over the popularity of C++ since it is so complex and presents an exceedingly complex algorithm for pairing teams in a round-robin tournament.

Yes, C++ is a complex language. Plauger is right in suggesting we choose and stick with a subset of the language. C++ is like English. You can use it to state something in a very complex fashion, or very simply. The latter is usually the stronger statement. The round-robin tournament problem illustrates this very well.

The physical-education community has developed a simple algorithm for pairing teams. First, list the teams on pieces of paper. Leave one piece blank, if necessary, to ensure an even number of teams. Lay out the papers in two rows; that's day #1. Hold the upper-left piece in place, and rotate all the other pieces of paper; you then have day #2. Repeat the process for each succeeding day until you are back at your original positions. For six teams, the rotation would be as in Figure 1.

The C++ program implementing this algorithm is equally straightforward. Note the simple, elegant power of the *for* statement controlling the inner print loop, which I show in Figure 2.

Jay Frederick Ransom
Oxnard, California

| | | | |
|---|---|---|---|
| Day 1: | 1 | 2 | 3 |
| | 6 | 5 | 4 |
| Day 2: | 1 | 6 | 2 |
| | 5 | 4 | 3 |
| Day 3: | 1 | 5 | 6 |
| | 4 | 3 | 2 |
| Day 4: | 1 | 4 | 5 |
| | 3 | 2 | 6 |
| Day 5: | 1 | 3 | 4 |
| | 2 | 6 | 5 |
| Day 6: | Repeats Day 1 | | |

***Figure 1***

```
#include <iostream.h>
#include <string.h>

char teams [100] [50],     // 100 teams ought to cover the field
                           // long names not allowed
     temp [50];
int   n_teams = 0,
     first_half,
     second_half,
     day,
     rotate;
char last_name;

main ()
{
     //  Read in team names
     do
     {
          cin >> teams [n_teams];
          last_name = teams [n_teams]  [0];
          n_teams++;
     } while (last_name  != '*');  //Delete the * team:
     n_teams -= n_teams % 2;       //Another simple, yet strong, expression

    //  print out playing schedule
    for  (day = 1; day < n_teams; day++)
    {
         cout << "\n\nDay " << day << "\n\n";
         for (first_half = 0,  second_half = n_teams  -  1;
              first_half < second_half;
              first_half++,  second_half--) //What you can do with a for loop!
             cout << "   "
               << teams [first_half]
               << " vs "
               << teams [second_half]
               << "\n";

      //  Rotate teams for next day

        strcpy (temp, teams  [1]);
        for  (rotate = 2;  rotate < n_teams; rotate++)
             strcpy  (teams [rotate - 1], teams [rotate]);
        strcpy (teams [n_teams - 1], temp);
    }
      return 0;
}
```

***Figure 2***

### *Keep It Simple*

Dear *DDJ*,

I was glad to read Michael Swaine's "Programming Paradigms" (*DDJ*, November 1993), which gives Forth a plug, even in a lighthearted way. It's a far cry, though, from the good old days of a decade ago when *DDJ* annually had an entire issue devoted to Forth.

Part of the Forth Standards efforts are confounded because the creator of Forth, Charles Moore, doesn't believe in standards for Forth. Moore considers Forth to be a program-development environment that increases productivity by speeding up the programming cycle. To keep it simple, small, and speedy (KISSS), certain design decisions resulted in using postfix notation, threaded code for compiling, a dictionary to hold functions, separate stacks for data and return addresses, and a (usually emulated) stack-based processor. As a result, the Forth language is an outgrowth of the Forth system, rather than a construct in its own right.

Moore works mostly with embedded systems, and varies the basic Forth to match the hardware and program-design requirements. Forth then behaves like assembler. A few primitive words (functions, opcodes) are used to extend Forth to develop the data types and structures that particular program requires.

On the other hand, a large group of programmers want to use Forth in symbolic programs: word processors, spreadsheets, graphics programs. They believe a Forth language without a standard is Forth in chaos. Many come from a traditional-language background—Fortran, Pascal, C, and the like. They want Forth to look more like the language they are familiar with, so they push for CASE statements, local variables, more stacks, string functions, floating-point numbers, graphics functions, and so on, as part of the standard Forth. It comes to this: Do you want Forth on a floppy or a CD-ROM?

Walter J. Rottenkolber
Mariposa, California

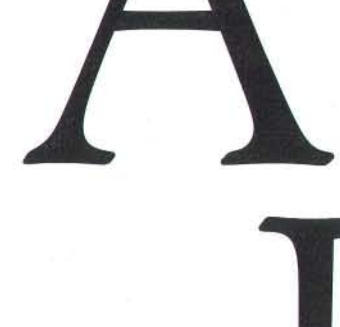
### *Windows Setup Follow Up*

Dear *DDJ*,

I wish that you had published Walter Oney's article, "Examining the Windows Setup Toolkit" (*DDJ*, February 1994) a few months earlier. Last November, I had the pleasure of building a setup program for an in-house software package using the setup toolkit from the Microsoft SDK.

# A Better Basic.

## New CA-REALIZER® 2.0 Beats Visual Basic 3.0.




Compared to Visual Basic 3.0, CA-REALIZER® 2.0 is half the cost and twice the product.

CA-REALIZER is already the easiest, most powerful BASIC for Windows and OS/2, and with version 2.0 it's even better. No one can match our combination of features, ease of use, and price. No one can make it easier to port your applications from QuickBASIC, and no one can make it more fun to develop for Windows and OS/2.

CA-REALIZER also comes with a huge array of powerful, plug-and-play tools like spreadsheets, charts, text editors, animation,



graphics tablets and database forms, along with many features other BASICs don't offer. Arrays are re-dimensioned and processed automatically. Algorithms can be written as formulae instead of complex looped expressions. Once an application is complete, compile it into a stand-alone OS/2 or Windows application and distribute it royalty-free with the run-time module that's included. And you can generate an installation disk with the push of a button.

*PC Computing* said, "No other Windows BASIC can match it for power and breadth of features."

At $99, no one can match our price either, and it includes our award-winning Windows report writer absolutely FREE.

### Only CA-REALIZER Gives You All Of This:

- Develop for Windows and OS/2
- Create and save libraries of pre-coded and formatted controls dynamically from within REALIZER
- Use any standard Windows and OS/2 custom control in FormDev
- Integrated Programmable Application Tools such as Charts, Spreadsheets, Text Editors, Graphics, Animation, and a Scheduler
- Easy migration of QuickBASIC applications to Windows and OS/2
- Import/Export Of 1-2-3, Excel, CA-Supercalc, CA-Compete! & Xbase Files
- Interactive WYSIWYG application design
- Generation of user accessible & modifiable BASIC code
- Full support for DDE and DLLs
- Instant database forms from DBF files
- Dynamically expandable multi-dimensional arrays with a full range of array and matrix operators and functions

Call 1-800-225-5224, Dept. 25302 Today Or See Your Local Dealer.

Microsoft Windows. Compatible

This offer is good for a limited time only. Call today and find out how much better BASIC can be.

Computer Associates® Software superior by design.

## New CA-REALIZER 2.0

CIRCLE NO. 803 ON READER SERVICE CARD

*(continued from page 10)*

I heartily agree with Walter: The setup toolkit is a credible toolkit (and free for SDK owners), but its documentation leaves you wishing for more.

There is one point in Walter's article which I can simplify. He recommends hand-modifying the .INF file produced by the DSKLAYT program. DSKLAYT is used to lay out the files to best fit on the setup disks. This program requires that you specify all of the files that will reside on the setup disks. This includes not only your applications files but the files that control the setup process. The .INF file, which is generated by the DSKLAYT program, controls the copying of the files from the setup disks to the user's hard disk. Normally, all of the setup files will also be listed in the .INF file and copied to the user's hard disk. Walter suggests removing these files from the .INF file by hand.

A better approach is to specify that the setup files be placed in a different section of the .INF file. This is controlled by the DSKLAYT program by filling in the "Put In Section" entry for each of the setup files. When this field is left blank, the files are put into the default section named "Files." I specify that all the setup files be placed into the section *SetupFiles*. When your setup script runs, it calls the function *AddSectionFilesToCopyList* and specifies the section name to add. For simple installation, only one call specifies the section "Files."

This is also the procedure to use when you want to allow the user to selectively setup portions of your application. You group your files in sections and allow the user to select which sections to setup. It is then a simple matter to call *AddSectionFilesToCopyList* for each section that is to be setup.

Gene Psoter
Atascadero, California

***Random Thoughts on the Stock Market***

Dear *DDJ*,

Tom Swan's "Algorithm Alley" column (*DDJ*, December 1993) correctly points out that a group of numbers, which are alleged to be random, must satisfy a lot of tests. But the highlighted example—the stock market—is not a good random sequence. The market is somewhat unpredictable. But successive prices are very strongly correlated. The distribution of the first differences (daily changes) is very uneven with far too many very small changes. You'd quickly scrap a random-number generator that created numbers like that. The other example, lottery numbers, is fine.

Donald Kenney
CompuServe 72630,3560

*Tom replies:* Thanks for your letter, Donald. You're right that successive stock-market prices are very strongly correlated, but so are the sequences produced by common random-number generators. In fact, so-called "random numbers" are completely predictable to produce the same sequence—just rerun the program using any value in the sequence as the starting seed! Stock market prices are more random because they are truly unpredictable. If that were not so, as I stated in the article, everyone would be wealthy.

I understand your point that stock-market prices themselves would not be suitable as direct substitutes for common random-function output, but I never said they were. If you use the Dow Jones Industrial Average to program a lunar-lander simulator, the landing module may crash. (Let's hope the stock market doesn't.) Seriously, I am not suggesting using stock prices as random numbers; only that the behavior of the stock market is an example of true randomness. Random-number generators are misnamed because their output is predictable, and therefore, not actually random. A generator's output may appear to satisfy some conditions of randomness, but only real-world events are truly chaotic.

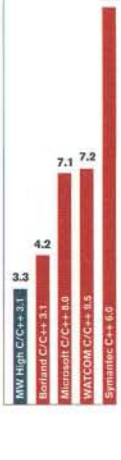
*(continued on page 16)*

MAKE YOUR CODE FLY

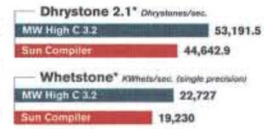
When you're pressing the limits of software performance, every microsecond counts. Veering from the optimum compilation course increases execution time.

That's why professional C/C++ developers use High C/C++™. MetaWare's High C/C++ compilers offer eight levels of global optimization, plus specific optimizations for particular processors. The executables consistently out-perform competitors in benchmark testing.

Besides offering rapid execution time, MetaWare's High C/C++ compilers provide a myriad of controls that allow you to customize and fine-tune code generation. There are command-line toggles and pragmas for customizing your application.




The compilers generate informative error and warning messages, and comply with major industry standards.




For years, MetaWare has supplied major OEMs such as AMD, AT&T, CenterLine, IBM, Intel, and NCR, with 32-bit compilers for a variety of different processors and platforms. These OEMs depend on MetaWare for superior products and support.

Call (408) 429-6382, or email: techsales@metaware.com, for more information about features and specific platforms.

**MetaWare**®

*High C/C++ for OS/2, NT, DOS, Windows, AIX, UNIX SVR4, and Solaris*

*Benchmark source and other performance information available upon request. Dhrystone benchmarks use the -O option.



**CIRCLE NO. 158 ON READER SERVICE CARD**

# Programmer's Paradise®

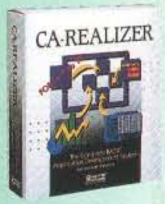
800-445-7899

## C++/Views 3.0
by Liant Software Corp.


The best object-oriented application framework for developing multi-platform, native GUI programs using C++. Includes a library of 100+ C++ classes that solve a broad range of software development problems including interface design, data management, event processing, & more. Includes C++/Views Constructor, a unique development tool that lets you work visually with the C++/Views class library. The Constructor unites an Interface Builder for creating portable resources & a Browser so you can switch from drawing & archiving your portable resources to editing the code that calls these resources. No royalties or RT fees. Source code is free.

**List: $749 *Special* Ours: $565 FAX*cetera* #: 1952-0021**

## CA-REALIZER for Windows & OS/2
by Computer Associates




Defines a new generation of development tools that handles the mechanics of event-driven programming, message passing, process sharing and other complexities behind the scenes. Combines a structured superset of BASIC extended to access Windows and OS/2 objects and resources, a visual development tool and Programmable Application Tools. CA-Realizer will help you create spreadsheets, charts, text editors, animation, graphics tablets and user-friendly forms from tools that can be created and manipulated by simple commands.

**List: $295 Ours: $79 FAX*cetera* #: 1004-0008**

## WindowsMAKER Professional 5.5
by Blue Sky Software

NEW VERSION!


If you're serious about C/C++ Windows development, WindowsMAKER™ Professional 5.5 is the most powerful Prototyper and C/C++ Code Generator for Windows, Win32 and Windows NT. This award-winning product offers more functionality & ease-of-use than any other tool. Create full-featured Windows Applications: MDI, Toolbars, Status bars, Templates, On-line Help, Graphical 3D buttons, Edit During Preview & much more. TrueCode™ technology ensures that user code is 100% preserved during code regeneration. Supports ANSI C, MFC C++, OWL C++ & more. Uses Switch-It™ Code Generation Modules for generating code for specific platforms, allowing migration between languages, C++ libraries & platforms. Highly recommended!

**List: $995 Ours: $875 FAX*cetera* #: 2602-0003**

## PRODUCT OF THE MONTH

## Borland C++ 4.0
by Borland International


New Borland C++ 4.0 is visual, and that's just the beginning. Only Borland C++ has true C++ support with exceptions, templates and the easiest, most powerful suite of object-oriented tools available today. With full support for DOS, Windows, Win32s and NT, it's the C++ development system you've got to have.

| | | |
|---|---|---|
| | **List: $499** | **Ours: $329** |
| **Upgrade** | **List: $199** | **Ours: $189** |

**FAX*cetera* #: 1861-0016**

## c-tree Plus®
by FairCom

DOS • WINDOWS • NT • UNIX • OS/2 • SUN • RS6000 • HP9000 • MAC • QNX • BANYAN • SCO. This well known, highly portable data management package has become established as the tool of choice for commercial development. Offering unprecedented data control, programmers may choose from direct low level access, ISAM level convenience, or SQL access with the FairCom Server. Single User, MultiUser, or Client/Server, ANSI Standard.

FAIRCOM® since 1979

**List: $595 Ours: $505 FAX*cetera* #: 1381-0008**

**Call Programmer's Paradise® *Italia* for special pricing in Europe.**

## IntegrAda for Windows
by Aetech Inc.


Complete Ada Programming Support Environment (APSE) for Windows, adding a language sensitive editor; automatic code generation & error checking; interactive error correction; pop-up menus to completely customize the environment to the standard Windows features. Allows you to quickly code, correct, compile, & link your Windows programs using Ada. Includes environment, libraries, validated Ada compiler, Ada Windows Libraries, Header Files, Resource & Help compilers, & MS Linker. (ACVC 1.11 validation pending). No RT Royalties!

**List: $995 Ours: $895 FAX*cetera* #: 2358-0003**

## Janus/Ada 9X Professional Development System
by R.R. Software

New! Ada 9X for the PC

The leader in PC-based Ada, proudly introduces the first comprehensive Ada 9X programming systems. Ada 9X brings the latest in Object-Oriented programming to Ada, and you can get it today complete with numerous professional tools. Special introductory pricing is good only until July 31, 1994.

**MS Windows NT or 32-Bit DOS Extender:**
**List: $995 Ours: $625 FAX*cetera* #: 1876-0001**

## Symantec C++ Professional 6.1
by Symantec


The new version 6.1 not only enhances product stability and reliability but also brings new features that 6.0 customers asked for. Now you get: full template debugging; improved hierarchical project manager; customizable color-syntax highlighting; & enhanced 32-bit support with 32-bit MFC 2.0 on the CD-ROM. So don't wait. Try Symantec C++ 6.1 and find out why the critics are raving about this new breakthrough in programming systems.

**Competitive Upgrade for Borland or Microsoft customers $189.**

| | | |
|---|---|---|
| | **List: $499** | **Ours: $299** |
| **Comp. Upg.:** | **List: $199** | **Ours: $189** |

**FAX*cetera* #: 2132-0038**

Programmer's Paradise®
Italia
Phone: 39-2-480-16053
FAX: 39-2-480-16039

# Best Sellers

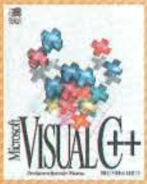
Call for a FREE Catalog!


**Network C Library**
by Automation Software Consultants, Inc.

NEW VERSION 3.0!

The most comprehensive library available for NetWare software development, supporting all versions of advanced NetWare. Over 450 C functions, include any features from the NetWare command line utilities and menu utilities in your C or BASIC programs for Windows or DOS. No licensing hassles. C library source code available.

**List: $395** **Ours: $355** **FAX*cetera* #: 1004-9201**




**Microsoft Visual C++ Development System v1.5**
by Microsoft Corporation

Master the power of OLE 2 & the database access flexibility of ODBC with MS Visual C++ v1.5, Prof. Edition. V1.5 allows you to create high-performance database apps & powerful reusable OLE 2 components. Includes Visual Workbench, App Studio, and MS Foundation Class Library 2.5.

| | | |
|---|---|---|
| **MS Visual C++ 1.5** | **List: $599** | **Ours: $379** |
| **w/Chinon CD-ROM Drive** | **List: $749** | **Ours: $699** |
| **1.5 Upgrade** | **List: $ 99** | **Ours: $ 89** |
| **w/Chinon CD-ROM Drive** | **List: $519** | **Ours: $479** |
| **Visual C++ 16/32 bit Suite** | **List: $199** | **Ours: $185** |

**FAX*cetera* #: 1269-0056**

**WATCOM FORTRAN 77[32] Version 9.5**
by WATCOM


32-bit optimizing FORTRAN 77 compiler and tools for DOS, Novell NLM, OS/2 2.x, Windows NT, Win32s, & Windows 3.x. Comprehensive language support with FORTRAN 90, DEC VAX & IBM VS language extensions. Advanced processor optimizations including 486 & Pentium instruction scheduling. Multi-platform toolset includes linker, debugger, profiler, royalty-free DOS extender with VMM & more.

**List: $599** **Ours: $349** **FAX*cetera* #: 1683-0005**

**RoboHELP® 2.6**
by Blue Sky Software


RoboHELP® 2.6, the best-selling Help Authoring Tool for Windows & Windows NT, offers full document to Help system conversion & vice versa. Turns Word for Windows into a fully functional hypertext authoring system capable of producing Windows Help files as easily as it does plain text. Fill in the actual Help text when prompted. RoboHELP takes care of generating the RTF, HPJ & H files. Link tester allows you to simulate your design before you compile. Full support of Word 2.0 & Word 6.0, & all features in the Windows Help Engine, such as macros, secondary windows, & multiple hotspot graphics.

**List: $499** **Ours: $439** **FAX*cetera* #: 2602-0005**

## NEW THIS MONTH

**ReferencePoint Personal Assistant**
by ReferencePoint


ReferencePoint Personal Assistant automatically makes an index of any information that you create or modify anywhere on your computer system, so that you can find any file, instantly, whether from a word processor, database or spreadsheet. So, type what you want to find, and Personal Assistant will locate every file in the computer containing those words. ReferencePoint Royalty Free SDK for DOS, Windows, Mac, OS/2 and UNIX also available from Programmer's Paradise®. Call for pricing.

**List: $99** **Ours: $ 79** **FAX*cetera* #: 1008-5001**

**WATCOM™ C/C++32 v9.5**
by WATCOM


C/C++32 is a professional, multi-platform C and C++ development system supporting 32-bit extended DOS, OS/2 2.x, Windows 3.x, Windows NT, Win32s, and AutoCAD ADS/ADI. The complete toolset includes: C and C++ optimizing compilers, royalty-free DOS extender with VMM support, licensed components from the MS Windows 3.x SDK, interactive source-level debugger, linker, profiler, supervisor for executing 32-bit applications and DLLs under Windows 3.x, 32-bit run-time libraries for extended DOS, OS/2 2.x, Windows 3.x and Windows NT, and more.

**List: $599** **Ours: $349** **FAX*cetera* #: 1683-0003**


**Mwave™ Developers Toolkit with IBM WindSurfer™ Bundle**
by Intermetrics, Inc.

What once was multiple products and toolsets is now bundled in one multimedia PC add-in card and Toolkit. The Mwave WindSurfer Communications Adapter is a "works out of the box" data/FAX modem, sound, voice messaging and telephone answering card with application software. It is bundled with the Mwave Developers Toolkit so you can build software that takes advantage of the Mwave digital signal processing (DSP) platform that drives WindSurfer. Try out the magic of a software upgradeable and programmable Mwave PC solution. You'll be developing for the future and using it today! And you can do it for an incredibly low price!

**List: $900** **Ours: $495** **FAX*cetera* #: 1012-4601**

**GUARANTEED BEST PRICES!** (Call for Details)

**To order call: 800-445-7899**
Corporate (CORSOFT): 800 422-6507
FAX: 908 389-9227
International: 908 389-9228
Customer Service: 908 389-9229
Programmer's Paradise Italia: 39-2-480-16053

For more information on the products featured on these pages call FAX*cetera*®: (201) 762-1378

Programmer's Paradise®
1163 Shrewsbury Avenue
Shrewsbury, NJ 07702

• All prices are subject to change without notice.
Call for details on return policy and shipping charge.

CIRCLE NO. 195 ON READER SERVICE CARD

800-445-7899

(continued from page 12)

### Putting HVC Back in Order

Dear *DDJ*,

In Maxwell T. Sanford II's letter (*DDJ*, November 1993) about my sidebar, "Putting Colors in Order" (*DDJ*, July 1993), Maxwell was too restrictive in assigning *Cmax=min(V, 1−V)*. It is true that R, G, and B must be in the range (0.0, 1.0), but the Chroma C can be as large as 2/3. This is true, for example, when V=2/3 and H=0. My new paradigm for representing a color is a color bubble, not a color cone. Given values for H and V, the max C is given by the procedure *GetMaxC* in Figure 3.

Harry J. Smith
Saratoga, California

```
procedure GetMaxC(H, V : Double;
                   var C : Double);
  { Get Max C for given H and V }
const
  Pi    = 3.14159265358979324;
  Pi2o3 = 2.0 * Pi / 3.0;
  DpR   = 180.0 / Pi; { Degrees per
                              Radian }
var
  HRad : Double;
  C1, C2, C3 : Double;
begin
  HRad:= H / DpR;
  C1:= Cos( HRad + Pi2o3);
  C2:= Cos( HRad);
  C3:= Cos( HRad - Pi2o3);
  if (C1 > 0.0) then
    C1:= V / C1
  else
    if (C1 < 0.0) then
      C1:= (V - 1.0) / C1
    else
      C1:= 1.0;
  if (C2 > 0.0) then
    C2:= V / C2
  else
    if (C2 < 0.0) then
      C2:= (V - 1.0) / C2
    else
      C2:= 1.0;
  if (C3 > 0.0) then
    C3:= V / C3
  else
    if (C3 < 0.0) then
      C3:= (V - 1.0) / C3
    else
      C3:= 1.0;
  C:= C1;
  if (C2 < C) then  C:= C2;
  if (C3 < C) then  C:= C3;
  if ((V <= 0.0) or (V >= 1.0))
     then  C:= 0.0;
end; { GetMaxC }
```

**Figure 3**

### Processor Scenarios

Dear *DDJ*,

Even after many months, I found Nick Tredennick's article, "Computer Science and the Microprocessor" (*DDJ*, June 1993) very interesting. His market model and definitions make a lot of (common) sense. During the course of the article, there is a recurrent theme of RISC vs. CISC and PC software vs. workstation software, and it becomes apparent that Nick feels that the two will never mix.

However, emerging software technology is providing the inevitability that the two worlds will mix. Without going into its relative merits, Microsoft's Windows NT is one of the first to desegregate hardware systems. NT is already on several microprocessors, and the list is likely to grow. This can only serve to help the microprocessor market waters find their own level. It should mean that the largest market share (CISC) will drop some, and the lowest market share (RISC) will rise.

The ultimate scenario is that any microprocessor house can provide a microkernel driver for their product and "Intel Inside" will have no more meaning than "GE Inside" for toaster-heating elements (although I've heard GE is considering using the Pentium in its toasters as heater elements). Then we can all pick our own favorite processor for the job at hand. Aaron Goldberg, in the September 27, 1993 *PC Week*, called this the "Esperanto of Tomorrow" (and showed why a multiplatform operating system has value, whereas Esperanto failed for lack of interest).

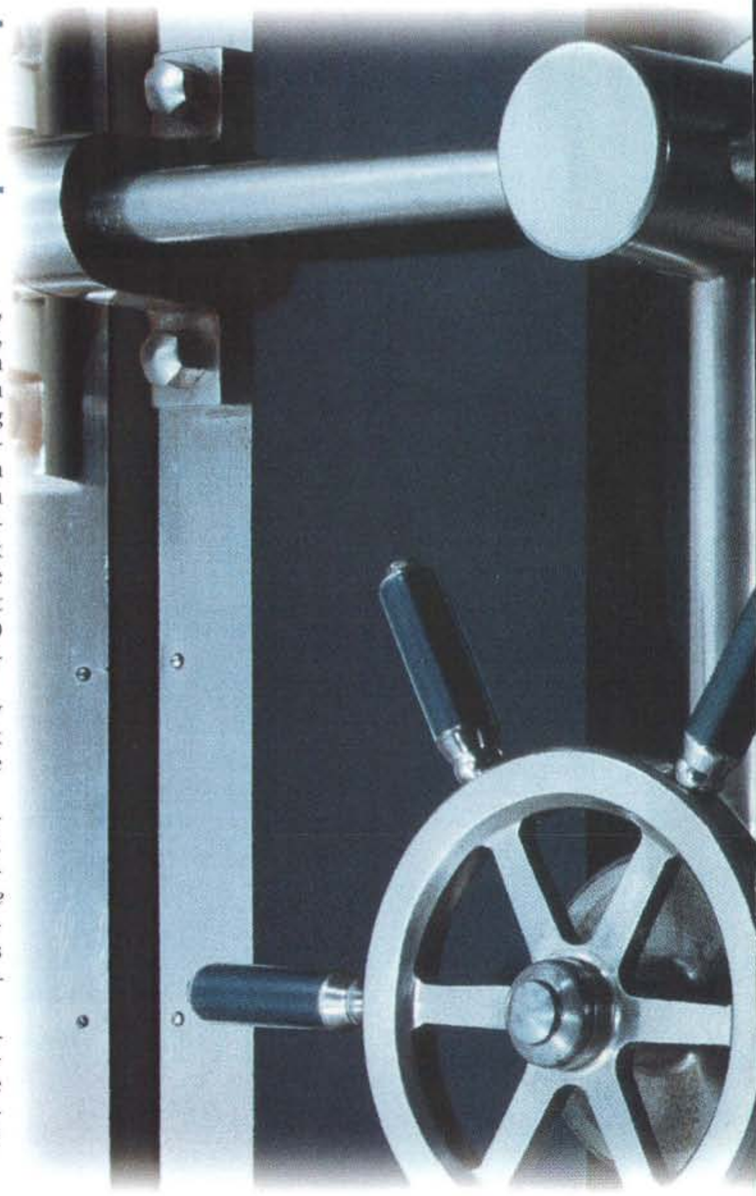
Jonathan Platt
Pipersville, Pennsylvania

**DDJ**

# The Cambridge Algorithms Workshop

*Designing fast, secure algorithms for high-performance applications*

Bruce Schneier



In December 1993, the Cambridge Algorithms Workshop, hosted by the Cambridge University Computer Laboratory, brought together leading figures in the field of encryption who presented, examined, and challenged new encryption algorithms designed to run quickly in software. The reasoning behind the focus on encryption is that constructing algorithms which are both fast and secure is the core problem of classical cryptology. However, recent developments, such as differential attacks on block ciphers and correlation attacks on stream ciphers, have forced cryptanalysts to rethink classic encryption algorithms such as those in Table 1. At the same time, the need for fast, efficient, and safe encryption at the application level has increased—DES (even triple-DES) may be fast enough for e-mail, but it's too slow for emerging high-bandwidth applications.

The goal of the conference, therefore, was to propose new algorithms capable of encrypting data at a dozen or so clock cycles per byte for the emerging class of high data-rate applications.

In this article, I'll cover the conference highlights and examine the current state of encryption technology in general. For specifics on the workshop, the proceedings will be published later this year by Springer-Verlag as part of their *Lecture Notes in Computer Science* series. (Call 201-348-4033 for availability.) Additionally, many of the topics and algorithms touched upon in this article are discussed in my book, *Applied Cryptography* (John Wiley & Sons, 1994).

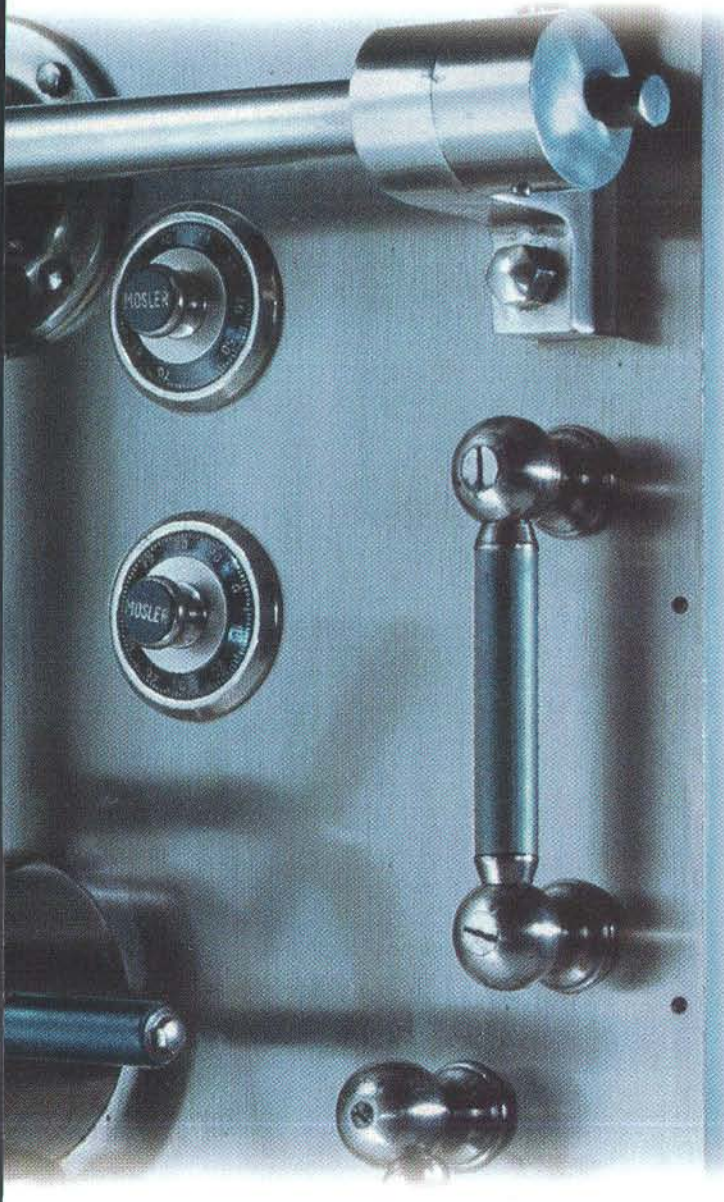
*Bruce, an independent software developer and author of* Applied Cryptography: Protocols, Algorithms, and Source Code in C *(John Wiley & Sons, 1994), presented a paper at the Cambridge workshop. Bruce can be contacted at 708-524-9461 or schneier@chinet.com.*

### The Algorithms

In all, ten complete algorithms were presented at the Cambridge Algorithms Workshop, all secret-key, not public-key algorithms like Diffie-Hellman, RSA, and the U.S. Government's Digital Signature Standard (DSS). (For more on public keys, see "Untangling Public-Key Cryptography," *DDJ*, May 1992.)

Jim Massey presented SAFER, a 64-bit block cipher with a 64-bit key. It is designed for implementation on simple processors on smart cards and only uses addition and multiplication mod 256, and exponentiation and logarithms in the multiplicative group of GF(257). Although Massey originally developed SAFER for Cylink Inc., the company has decided to place this algorithm in the public domain in the hope that it will become the new standard for software encryption. This algorithm will probably be important. In fact, former Soviet cryptanalysts in Yerevan, Armenia have been—without success—trying to break SAFER for over a year.

Matt Robshaw of RSA Data Security presented a fast block cipher based on the same principles as the MD5 one-way hash function (see my article, "One-Way Hash Functions," *DDJ*, September 1991). Robshaw's algorithm, designed jointly with Burt Kaliski, operates on large blocks—1024 bytes—but the principles can be used to design ciphers with smaller block sizes. It is also likely to be important, as RSA Data Security provides encryption technology to companies such as Microsoft, Lotus, IBM, Novell, and Apple.




Phil Rogaway presented SEAL, a new stream cipher developed jointly with Don Coppersmith, IBM's top cryptographer. This algorithm will be used by IBM for software encryption of disk files in PC-based network software. The algorithm is based on repeated lookup of large tables of pseudorandom numbers.

Hugo Krawczyk of IBM presented implementation and performance details of the Shrinking Generator, which he also designed with Don Coppersmith and first presented at Crypto '93. (See "The Shrinking Generator," *Advances in Cryptology: CRYPTO '93 Proceedings*, Springer-Verlag, in preparation.) The Shrinking Generator is a simple stream cipher which uses the output of a linear-feedback shift register to decimate the output of another linear-feedback shift register.

Markus Dichtl of Siemens presented a derivative of the Shrinking Generator which combines the output of linear-congruential generators rather than linear-feedback shift registers.

Ross Anderson of Cambridge University presented a modern rotor machine. Rotor machines were all the rage in cryptography before algorithms moved to computers. The German Enigma was a rotor machine, as were several U.S. cryptography machines. They have fallen out of fashion recently, but this algorithm is an attempt to show that they can still be secure in the face of massive computing power. Anderson's proposal is for a rotor machine driven by a linear-feedback shift register; it is simple to describe and to implement, and yet seems to be very secure.

David Wheeler, also of Cambridge, proposed a bulk-encryption algorithm based on experience designing algorithms for secure digital telephones. It is based on iterating functions defined by large lookup tables of random permutations, and can perform ultra-fast encryption of large amounts of data.

My own Blowfish algorithm is a 64-bit block algorithm with a variable-length key. (See "Blowfish: A New Encryption Algorithm," on page 38 of this issue.)

William Wolfowicz (of the Italian telephone company's research lab) presented a 64-bit block algorithm with a 128-bit key which was designed jointly with Adina di Porto. The algorithm makes use of a novel permutation property pointed out by the Russian number theorist Vinogradov.

Joan Daemen presented 3-WAY, a new block cipher he's been developing with colleagues at Leuven, Belgium for two years. It is designed to be fast in both hardware and software, and to resist both differential and linear cryptanalysis attacks.

The fastest algorithm appears to be either Wheeler's or Rogaway-Coppersmith's. Both use about 20 instructions (including four table lookups) to encrypt a 32-bit word: about five clock cycles per byte. On a SparcStation, they had throughput of about 20 Mbytes/second; on a DEC Alpha, about 100 Mbytes/second. However, many of the other algorithms are not far behind.

It is too early to tell if any of these algorithms are secure. The important thing is that they are out there and that cryptanalysts are starting to examine them. I expect at least two of them to fall before the end of the year. (It would be unfair to divulge which I think are insecure.) The techniques used to break them will be recycled into the design of new encryption algorithms, which may then be broken using new techniques. And so the cycle of research will continue.

Hopefully, in five or six years there will be a few algorithms that are still considered secure. These may then be proposed as standards to replace DES and then used to encrypt data far into the next century.

### Designing Secure Algorithms

If nothing else, the Cambridge workshop proved that fast, efficient, and safe encryption algorithms are as difficult and challenging to design as ever. The rules of algorithm design are simple. An encryption algorithm should be secure under the following conditions:

- The cryptanalyst (that's the guy trying to break the algorithm) knows all the details of the algorithm. He has some ciphertext, and his job is to deduce the plaintext. (This is called a "ciphertext-only" attack.)
- The cryptanalyst not only has the algorithm and some encrypted ciphertext, but also the unencrypted plaintext. His job is to deduce the key. (This is called a "known-plaintext" attack.)
- The cryptanalyst not only has the algorithm, some ciphertext, and the unencrypted plaintext, but he gets to choose what it is. If there is a particular plaintext sequence that, if encrypted, will easily yield the key, he gets to encrypt that sequence. (This is called a "chosen-plaintext" attack.)

All of these attacks are feasible and have been mounted in the real world. (For historical anecdotes, see *The Codebreakers: The Story of Secret Writing,* by D. Kahn, Macmillan, 1967.) Often, noncryptographers insist that the details of their algorithm should remain secret. From the point of view of security, this is a dubious practice. Security should not depend on the secrecy of the algorithm. If it did, it would be far too vulnerable to a "black bag" attack. A hardware-encryption device can be stolen and reverse-engineered; a software-encryption device can be disassembled. (Even the details of DES were published by the government; see the National Bureau of Standards, Data Encryption Standard, U.S. Department of Commerce, FIPS PUB 46, January 1977.) Cryptographers have to assume that analysts will have everything but the key, simply because it is prudent to do so.

The Skipjack algorithm remains classified and is implemented in the supposedly tamper-resistant Clipper chip. When Intel came up with a new reverse-engineering technique which they thought might beat the tamper protection, the NSA promptly classified it. Even so, the algorithm is probably resistant to anal-

| | Key Length | Block Length | Problems |
|---|---|---|---|
| DES | 56 bits | 64 bits | Key too small |
| Triple-DES | 112 bits | 64 bits | Slow |
| Khufu | 64 bits | 64 bits | Patented; key too small |
| FEAL 32 | 64 bits | 64 bits | Patented; key too small |
| LOKI-91 | 64 bits | 64 bits | Weaknesses; key too small |
| REDOC II | 160 bits | 80 bits | Patented |
| REDOC III | variable | 64 bits | Patented |
| IDEA | 128 bits | 64 bits | Patented |
| RC2 | variable | 64 bits | Proprietary |
| Skipjack | 80 bits | 64 bits | Secret algorithm |
| GOST | 256 bits | 64 bits | Not completely specified |
| MMB | 128 bits | 128 bits | Insecure |

***Table 1:** Block-encryption algorithms. IDEA is used for message encryption in Pretty Good Privacy; PGP. RC2 was developed by RSA Data Security and is used in a variety of commercial software packages. Skipjack is the NSA-developed algorithm in the Clipper chip. GOST is an algorithm developed in the Soviet Union and only recently made public.*


## Professional C and C++ Development Tools

C/C++$^{32}$ delivers the key technologies for professional developers: comprehensive C++ support including templates and exception handling; advanced superscalar optimization; and 32-bit multi-platform support. C/C++$^{32}$ includes both C and C++ compilers, so you can incrementally adopt the benefits of C++.

## Unleash 32-bit Power!

C/C++$^{32}$ delivers 32-bit performance. The 32-bit flat memory model simplifies memory management and lets applications address beyond the 640K limit. Powerful 32-bit instruction processing delivers a significant speed advantage: typically a minimum 2x processing speedup.

## A C++ compiler designed to deliver on the promise of object-oriented programming

The C++ compiler provides comprehensive support for the AT&T version 3.0 language including templates, plus exception handling. These features are key to realizing the benefits of object-oriented programming: code reusability, increased reliability and reduced maintenance.

## Hot New Superscalar Code Optimizer

The hot, new C/C++$^{32}$ code generator advances the performance envelope. New superscalar optimization strategy uses "riscification" and instruction scheduling to deliver improved performance on 486 and Pentium processors. The compiler can create a single, high-performance executable which runs on 386, 486 and Pentium processors.

## Industry Standard. Industry's Choice

WATCOM's working relationships with industry leaders such as Autodesk, GO, IBM, Intel, Lotus, Microsoft and Novell ensure that we continue to understand and meet the needs of the software industry and professional developers.

## Multi-Platform, Cross Development Support

C/C++$^{32}$ supports a wide range of 32-bit Intel x86 host and target platforms allowing professional developers to leverage the multi-platform, cross-development capabilities of today's operating environments including OS/2 2.x and Windows NT.

- **32-bit DOS** host and target support; includes the DOS/4GW 32-bit DOS extender by Rational Systems with royalty-free run-time and virtual memory support up to 32Mb.
- **OS/2 2.x** host and target support; enables the development of OS/2 2.x applications and DLLs. 32-bit Windows 3.x applications can exploit WIN-OS/2 using WATCOM's Windows Supervisor technology.
- **Windows NT** host and target support; includes a Windows NT-hosted debugger for 16-bit Windows 3.x, Win32s and Windows NT debugging; also includes **Win32s** target support.
- **32-bit Windows 3.x** target support; includes WATCOM's 32-bit Windows Supervisor, and enables development and debugging of true 32-bit Windows 3.x applications and DLLs.

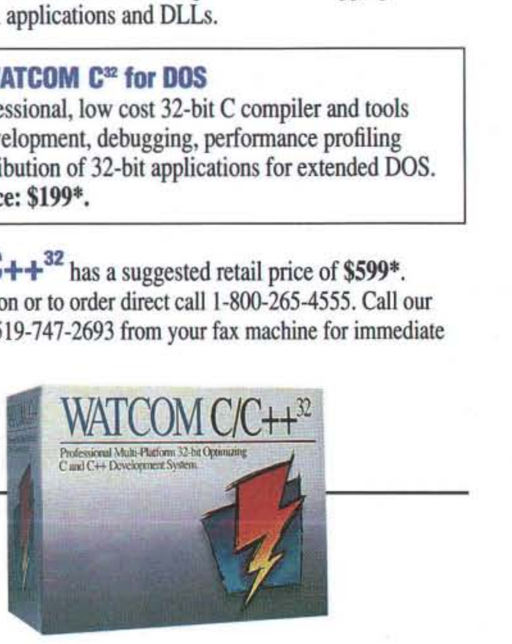
### Also Available: WATCOM C$^{32}$ for DOS

C$^{32}$ for DOS is a professional, low cost 32-bit C compiler and tools package enabling development, debugging, performance profiling and royalty-free distribution of 32-bit applications for extended DOS. **Suggested retail price: $199*.**

**WATCOM C/C++$^{32}$** has a suggested retail price of **$599***. For additional information or to order direct call 1-800-265-4555. Call our FAX Back system at 1-519-747-2693 from your fax machine for immediate product information.

CIRCLE NO. 698 ON READER SERVICE CARD

WATCOM 1-800-265-4555




## The Leader in 32-bit Development Tools

415 Phillip Street, Waterloo, Ontario, Canada, N2L 3X2 Telephone: (519) 886-3700, Fax: (519) 747-4971

*Price in US dollars. Does not include freight and taxes where applicable. Authorized dealers may sell for less. WATCOM C and the Lightning Device are trademarks of WATCOM International Corp. DOS/4G and DOS/16M are trademarks of Rational Systems Inc. Other trademarks are the properties of their respective owners.

*(continued from page 20)*
ysis even if its details become known. (The primary reason that NSA does not want to release the inner workings of Skipjack is probably because they don't want their secret techniques used to design other high-security algorithms.)

Cryptographers also have to assume that analysts have access to enormous amounts of computing power—more computing time than can be optimistically expected during the next 100 years or so. In the face of all these conditions, the algorithm should still be unbreakable.

*The goal of the conference was to propose algorithms capable of encrypting data at a dozen or so clock cycles per byte*

### Key Length

Key length is a poor measure of the security of an algorithm, but it's a good place to start. Algorithms with long keys are not necessarily secure, but algorithms with short keys are definitely insecure.

For instance, earlier this year Michael Weiner presented a design for a brute-force DES cracker (see "Efficient DES Key Search," *Advances in Cryptology: CRYPTO '93 Proceedings,* Springer-Verlag, in preparation). He didn't just do some theoretical calculations, but went through the entire design process. He designed a custom cracking chip down to the gate level and had his in-house fabrication department estimate fabrication costs. He designed a controller board, had its cost estimated, and designed and priced peripheral hardware (power supplies, racks, and a complete mechanical housing). What Weiner determined was that with a $1 million machine, he could break DES in three-and-a-half hours. This is cheap enough to hide in the budget of several different government agencies. It's even cheap enough to be considered by large corporations or organized-crime syndicates. (His employer, Bell Northern Research, claims to have no interest in building a working model.)

This work has important implications for algorithm design. There's nothing special about DES; the analysis will be similar for all algorithms. 56 bits is too small for a key; even 64 bits is too small. Even 80 key bits is marginal—only enough for short-term security. Any algorithm proposed today should have a key length of at least 128 bits; see Table 2.

These calculations are based on present-day computers. For future projections, plan on computing power doubling every 18 months. Each of the above numbers becomes an order of magnitude smaller every five years: A $1 billion machine that takes 6.7 years to break a key today will take 0.67 years (8 months) with 1999 technology and 0.067 years (24 days) with 2004 technology. What is secure now might not be in 50 years. In light of these calculations, Skipjack's 80-bit key seems woefully inadequate.

Key length is also critical for export. The U.S. Government does not permit the export of algorithms with key lengths greater than 40 bits. (Yes, some exportable algorithms appear to have longer key lengths, but the effective key length is 40 bits or less.) Various computer-privacy advocates are trying to change this.

Some of the algorithms presented at the Cambridge workshop had variable-length keys. This is especially desirable because it allows the implementor to define his own level of security. If he has to export the algorithm, the implementor can set the key length at 40 bits. For low-grade security (information that only has to remain secret for a few minutes), you can use 64 bits. If you need long-term security, you can use key lengths of 128 or even 256 bits.

Variable-length keys were generally constructed during a "key-expansion phase" of the algorithm. Generally, there was an initial bit of computation required before the algorithm could encrypt any data. During this computation, the key typed in by the user would be expanded into a large set of subkeys used for encryption. DES does this to some degree; the 56-bit key is expanded into an array of subkeys totaling 768 bits. Some algorithms at the Cambridge workshop took this to an extreme, expanding a key into subkey arrays totaling 1 Kbyte of data or even more.

### Differential and Linear Cryptanalysis

Of course, the trick to algorithm design is to make sure that a brute-force attack is the most efficient way of getting the key, although for most encryption algorithms, there are other ways. These methods, generally very complex and mathematical, involve exploiting the structure of the algorithm.

Differential cryptanalysis and linear cryptanalysis are two new attacks that have been successfully used against DES and other algorithms. Differential cryptanalysis was invented by Biham and Shamir in 1991 (see *Differential Cryptanalysis of the Data Encryption Standard,* by E. Biham and A. Shamir, Springer-Verlag, 1993), while linear cryptanalysis was invented by Mitsuru Matsui in 1993 (refer to "Linear Cryptanalysis Method for DES," *Advances in Cryptology: CRYPTO '93 Proceedings,* Springer-Verlag, in preparation). Differential cryptanalysis is a chosen-plaintext attack: It looks at differences between pairs of plaintexts and corresponding pairs of ciphertexts. These differences, along with information about the structure of the underlying algorithm, give an analyst clues about the key. Collect enough of these differences, and you can find the key more efficiently than you would with brute force.

Don't get too excited, though. The best chosen-plaintext differential cryptanalysis attack against DES has a complexity of $2^{47}$. This is better than the $2^{56}$ required for brute force, but requires on the order of 10 terrabytes of chosen-plaintext data. Although interesting, it is still more theoretical than practical. The best way to attack DES is still brute force.

| Key Length | Time for a $1M Machine to Break | Time for a $1B Machine to Break |
|---|---|---|
| 40 bits | 0.2 seconds | 0.0002 seconds |
| 56 bits | 3.5 hours | 13 seconds |
| 64 bits | 37 days | 54 minutes |
| 80 bits | 2000 years | 6.7 years |
| 100 bits | 7 billion years | 7 million years |
| 128 bits | $10^{18}$ years | $10^{15}$ years |
| 192 bits | $10^{37}$ years | $10^{34}$ years |
| 256 bits | $10^{56}$ years | $10^{53}$ years |

***Table 2:*** *Key length and security in 1994.*

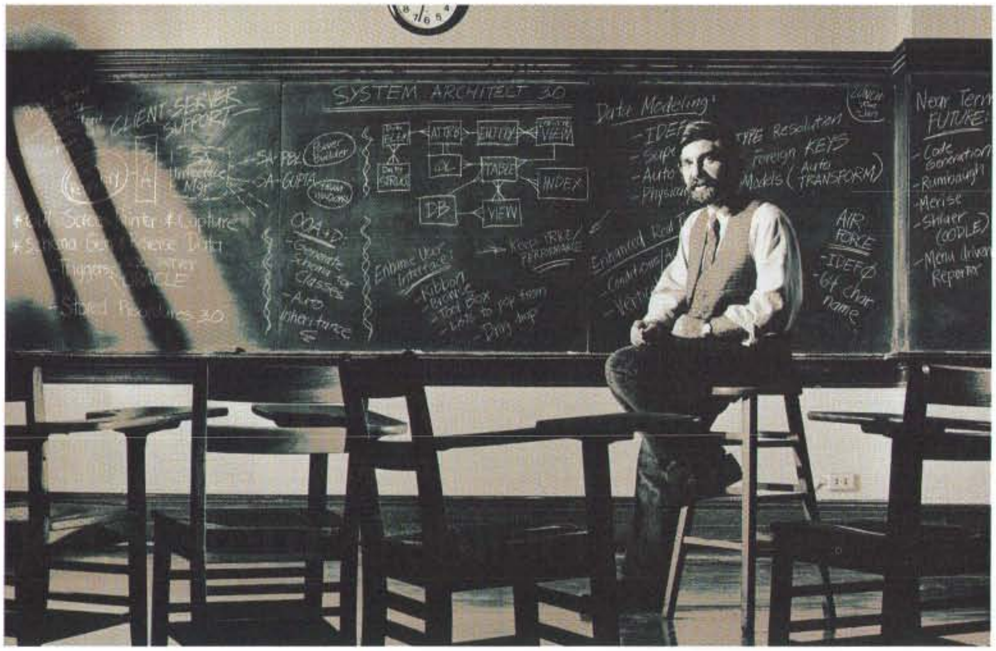
| Attack | Type | Complexity |
|---|---|---|
| Brute-force | Known-plaintext | $2^{55}$ |
| Differential | Known-plaintext | $2^{55}$ |
| Differential | Chosen-plaintext | $2^{47}$ |
| Linear | Known-plaintext | $2^{43}$ |

***Table 3:*** *Cryptanalysis of DES.*

# "You asked for a powerful and affordable tool to develop client/server applications. That's why I developed System Architect 3.0."

JAN POPKIN, CHIEF SCIENTIST
POPKIN SOFTWARE & SYSTEMS, INC.

Developers and project teams looking for a CASE-based tool for client/server application development will find the answer in System Architect™ 3.0. This latest version of the CASE price/performance leader includes the features of expensive tools for a fraction of the cost.



## System Architect 3.0.

SA 3.0 simplifies the development of client/server applications by supporting multiple methodologies including Information Engineering, Gane & Sarson, Yourdon, IDEF, OOA&D, SSADM IV, Shlaer/Mellor, and Ward & Mellor. It also features an integrated data repository you can customize. And it runs under MS Windows® or IBM's OS/2 PM.®

## Flexibility And Functionality.

The ideal combination of flexibility and functionality has made SA the undisputed price/performance leader. As the needs of developers have changed, so has the scope of SA's features and options:

MICROSOFT. WINDOWS.

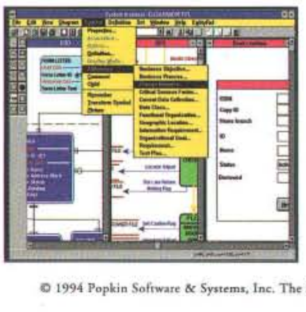
*SA Screen Painter:* Allows repository-based development of GUI screens and menus or character-based screens.

*SA Object-Oriented Version:* Supports Booch '91 and Coad/Yourdon.

*SA Reverse Data Engineer:* Reverse engineers SQL databases, including SQL Server, SYBASE, DB2, Informix, and Oracle.

*SA Schema Generator:* Generates DDL and SQL triggers from entity models for Oracle, Informix, Ingres, PROGRESS, Paradox, dBASE III, DB2, SQL Server, SYBASE, and other SQL and 4GL databases.

*SA Project Documentation Facility:* Enables the automatic generation of deliverables with desktop publishing quality from SA Encyclopedia.

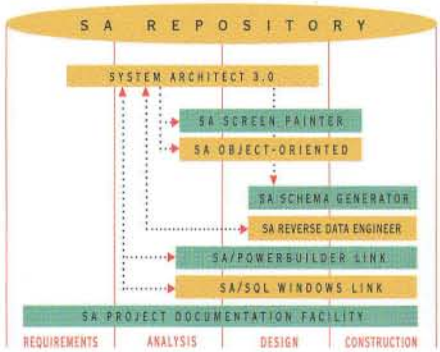


◄*Paint GUI screens from data in repository.*

## Choose Your Development Environment.

*SA/PowerBuilder Link:* Allows the exchange of design information between SA and PowerBuilder for the development of more robust client/server applications.

*SA/SQL Windows Link:* Works with Gupta's SQL Windows.




▲*System Architect 3.0 covers your development lifecycle with a complete range of features and options.*

## Put Your Project Team In A Class Of Its Own.

System Architect 3.0 makes your project team more productive with a range of capabilities including:

*Network Version:* Allows multiple team members to work concurrently on a project while sharing the SA Repository by locking diagram and data dictionary records.

*Network Security:* Allows Project Managers to uniquely identify and classify personnel with appropriate levels of authorization.

*Access Control:* Allows team members to check-out, check-in, or freeze encyclopedia objects with defined authorization.

*Version Control:* Allows project encyclopedias, and their related files, to be saved and stored with appropriate version-identifying data. (Available in version 3.1)

## Call Us Today At 800-REAL-CASE, X131.

To find out how to qualify for your free 30-day evaluation copy, call us today, or fax us at 212-571-3436.

SYSTEM ARCHITECT™

Popkin Software & Systems, Inc.,
11 Park Place, New York, NY 10007

England 44-926-881186; Benelux 31-3406-65530;
Germany 49-6151-82077; Italy 39-49-8700366;
Switzerland 41-61-6922666; Denmark 45-45-823200;
Australia 61-02-346499; Sweden 46-8-626-8100;
Elsewhere 1-212-571-3434



**CIRCLE NO. 321 ON READER SERVICE CARD**

*(continued from page 22)*

Linear cryptanalysis is similar to differential cryptanalysis, but looks for linear relationships between selected bits of the plaintext, ciphertext, and key. Against DES, this attack has a complexity of $2^{43}$. Even better, it is a known-plaintext attack. However, it still requires much too much data to be practical.

Now that these attacks are known, there are techniques for optimizing encryption algorithms so that they are resistant to them. Most of the algorithms presented at the workshop took these attacks into account during the design process; see Table 3.

### Cascading Multiple Algorithms

One way to increase the security of your system is to chain multiple algorithms together. For example, first encrypt your file with DES and one key, and then with IDEA (see "The IDEA Encryption Algorithm," by Bruce Schneier, *DDJ*, December 1993) and another key. The result will be much stronger than using either of the two algorithms individually.

Cascading multiple algorithms might also be the best way to negotiate security with algorithms that some people don't trust. If Alice and Bob want to communicate with each other and don't trust each other's algorithms, they can use both—first her algorithm, and then his. This idea, suggested by Whitfield Diffie, was discussed at the Cambridge workshop.

This sounds good, but there is a problem: Massey and Maurer proved that a cascade of multiple algorithms is at least as strong as the first or, with stream ciphers, at least as strong as the best (see "Cascade Ciphers: The Importance of Being First" by Maurer and Massey, *Journal of Cryptology,* 1993). The difficulty with proving anything more than this is that a bad guy might provide you with a first algorithm which twisted your plaintext around so as to provide a chosen-plaintext attack on the second algorithm which you supplied.

This applies not just to encryption algorithms, but to any process designed by someone else which you incorporate into your system. In fact, the widely used CELP code (which compresses digital speech to modem speeds) was designed by the NSA, and for all anyone knows it could be acting as a cryptanalyst's helper in some subtle way. It does seem though, that a cascade of algorithms is better than individual algorithms, provided that the second and subsequent algorithms are secure against chosen-ciphertext attacks, and provided all the algorithms' keys are independent.

The real benefit of cascading algorithms is in design diversity; it makes the overall system less vulnerable to a cryptanalytic attack. Both triple-DES and IDEA seem secure today, but there is always the possibility that some clever mathematician might come up with a good attack against one of them tomorrow. Using triple-DES, then a fast stream cipher such as Wheeler's algorithm, and then IDEA, would be immune to new attacks against any one of the three algorithms. Successful attacks against all three would be required to break the cascaded system.

### Conclusion

Encryption algorithms are like airplanes. It's easy to design one, but it's hard to design one that flies. To make matters worse, it's hard to tell if any one of them is any good. The only real way to test the security of an algorithm is to let other programmers try breaking it. But even if the algorithm survives years of intense analysis by many different people, you can still only hope that it is really secure.

**DDJ**

To vote for your favorite article, circle inquiry no. 1.

# One source is all you really need.


The news today is the Wind/U 2.0™ portability toolkit! Wind/U enables your Microsoft Windows and Visual C++ applications to run as native UNIX/Motif applications. You can use your Microsoft Windows source base to rapidly penetrate the fast growing workstation marketplace.




The advantage of Wind/U is that the toolkit uses the complete set of the Microsoft Windows application programming interface (API) facilities under Motif. You only need to recompile your Windows application in the UNIX environment and link to the Wind/U Library. Wind/U maps the Windows function calls to Motif and X calls, allowing your Windows application to execute as a native Motif application.

Wind/U ported applications maintain identical functionality between Windows and Motif versions since Wind/U provides features such as on-line help, PostScript and PCL printing, DDE, MDI, and Common Dialogs. Wind/U supports Win16, Win32s, and the Microsoft Foundation Class Library.

Now you can cost-effectively run your applications on Sun, IBM, Hewlett-Packard, and DEC workstations, while focusing your programming efforts on the popular Windows API.

wind U™

*For more information call (203) 438-6969, email to info@bristol.com or fax to (203) 438-5013.*

**Bristol Technology Inc.**

*Bringing the Best of Microsoft Windows to X and Motif*

**CIRCLE NO. 851 ON READER SERVICE CARD**

# Cryptography Without Exponentiation

## *Secure alternatives to RSA*

Peter Smith

While not a full-fledged public-key cryptosystem, the 1976 Diffie-Hellman key-negotiation technique featured the first cryptographic use of modulus exponentiation. Diffie and Hellman's method, which is used to establish a secret key over an insecure channel, is still in use because the mathematical problem on which it is based remains as difficult today as it was in 1976. In this, Diffie and Hellman were also the first to base cryptosystems on problems that mathematicians have been unable to solve.

In the January 1993 issue of *DDJ*, I presented an alternative to the RSA encryption algorithm called LUC (see "LUC Public-Key Encryption," *DDJ*, January 1993). As that article suggests, many ciphers, including the Hellman-Diffie-Merkle key-exchange system and the El Gamal digital signature, can be reinforced by replacing the process of exponentiation with the process of calculating Lucas functions. This article extends LUC with three new cryptosystems: the Lucas-function El Gamal public-key encryption, the Lucas-function El Gamal digital signature, and a Lucas-function-based key-negotiation method called LUCDIF.

*Peter has worked in the computer industry for 16 years, and served as deputy editor of* Asian Computer Monthly. *He invented LUC in 1991 and founded LUCENT to commercialize Lucas-function-based cryptography. He can be reached at 25 Lawrence Street, Herne Bay, Auckland, New Zealand.*

### The Algorithms

The exponentiation ciphers here are all based on the mathematical problem known as the Discrete Logarithm (DL). Basically, this problem reduces to solving for x in the equation $a^x$=b mod c; where a, b, and c are integers and their values are known. The cipher known as El Gamal and its variants were introduced over the course of the 1980s and are based on the DL problem. One of these, Schnorr's variant of the El Gamal digital signature, was chosen by the National Institute of Standards and Technology as the basis of the Digital Signature Standard.


As suggested in my previous article, ciphers based on the DL problem can be implemented using Lucas functions instead of exponentiation. Such implementations are sometimes not without their complications in terms of storage and timing overheads, but they can be shown to be asymptotically as fast. More importantly, they are cryptographically stronger than their exponentiation-based ancestors. It is an open question how much stronger the Lucas-function ciphers are. The fastest known subexponential-time algorithms for attacking the DL can't be used against them, making them vulnerable only to exponential-time attacks.

The mathematical problem on which the Lucas-function ciphers are based is analogous to the DL problem, except that here the problem is to solve for x in the equation $V_x(a,1)$=b mod c. This problem has the advantage that the subexponential algorithms do not appear to generalize to it, so breaking these ciphers is much more expensive.

### Key Negotiation

The Diffie-Hellman key-negotiation process allows two correspondents, Alice and Bob, to establish a common cryptographic key between them, even if an eavesdropper is listening in on their connection. They both agree on a prime p and a primitive root (or generator) $\alpha$. Using a secret number A, Alice publishes her part of the key, as given by $\alpha^A$ mod p. Similarly, Bob publishes his part of the key using his secret number B, using the formula $\alpha^B$ mod p. In Alice's case, she takes Bob's key and forms $(\alpha^B)^A$ mod p, while Bob takes Alice's published key, and forms $(\alpha^A)^B$ mod p. Since $(\alpha^B)^A$ mod p equals $\alpha^{AB}$ mod p equals $(\alpha^A)^B$ mod p equals some value K, say, then both Alice and Bob now have the same key. This method, the first successful, though partial, implementation of public-key ideas, lets part of the key be made public.

The DL problem seems to guarantee that an eavesdropper, who has only public knowledge and not the secret values A and B, cannot find K. If p, A, and B are large enough (say, over 500 bits in length), there is only a small chance of guessing the secret values. If the key were to be used as a DES key, Alice and Bob could agree to take only the first 56 bits to K.

We have called our Lucas-function-based key-negotiation method LUCDIF, combining LUCas and DIFfie. As with LUC, the known multiplicative attacks on Diffie-Hellman do not carry over to LUCDIF, since it is not multiplicative.

LUCDIF is quite analogous to Diffie-Hellman. Choose the prime p in the same way. A value $\lambda$ must be chosen so

(from page 26)
...dition in Figure 1(a) is true. ...h a value is easy. Every val... ...as a 50 percent chance of sat...ing the condition. Now the values given by $V_A(\lambda,1)$ mod p and $V_B(\lambda,1)$ mod p are published by Alice and Bob, respectively. Bob takes Alice's number and calculates $V_B(V_A(\lambda,1))$ mod p. Similarly, Alice calculates $V_A(V_B(\lambda,1))$ mod p.

Relation 1 in Figure 1(b) shows that these two values are the same and that Alice and Bob have obtained the same key, K'. If p is a prime of over 512 bits, then this method of key negotiation is very secure. Once again, for a DES key, Alice and Bob may decide to select only the first 56 bits of K'.

(a) $V_{(p+1)/t}(\lambda,1)\neq 2 \bmod p$, for all $t>1$ dividing $(p+1)$

(b) $V_{nm}(P,Q)=V_n(V_m(P,Q),Q^m)$ Relation 1

$2V_{n+m}=V_nV_m+DU_nU_m$ Relation 2

***Figure 1:*** *(a) Choosing a value* $\lambda$*; (b) Lucas-function relations which let us transform exponentiation to Lucas-function calculation.*

### El Gamal and LUCELG

The El Gamal cipher comes in two parts. There is a procedure for encrypting and decrypting and a second procedure for signing and verifying a digital signature.

For encryption, assume Alice wants to send a message M to Bob using his public key y which is equal to $\alpha^x$ mod p (x is Bob's private key). Alice first finds a secret number k, which is greater than zero and less than p, and calculates L using $L\equiv y^k$ mod p. Two other values are then worked out: $c_1\equiv\alpha^k$ mod p, and $c_2\equiv LM$ mod p. These two values, $c_1$ and $c_2$, make up the cryptogram which Alice sends to Bob.

*Schnorr's variant of the El Gamal digital signature was chosen as the basis of the Digital Signature Standard*

For decryption, Bob first calculates L using the fact that $L\equiv(\alpha^k)^x=(c_1)^x$, since only he knows the value of his secret key x. Having found L, Bob calculates its multiplicative inverse $(L^{-1})$, and multiplies this by $c_2$, recovering M; $M\equiv c_2(L^{-1})$ mod p.

The Lucas-function version of El Gamal public-key encryption and decryption follows a path similar to that of El Gamal public-key encryption/decryption. Bob's public key, in this case, is $V_x(\gamma,1)$ mod p. A secret value k is also necessary here, and we first calculate G. When encrypting, $G\equiv V_k(y,1)$ mod p. The two halves of the cryptogram are then computed: $d_1\equiv V_k(\gamma,1)$ mod p, and $d_2\equiv GM$ mod p.

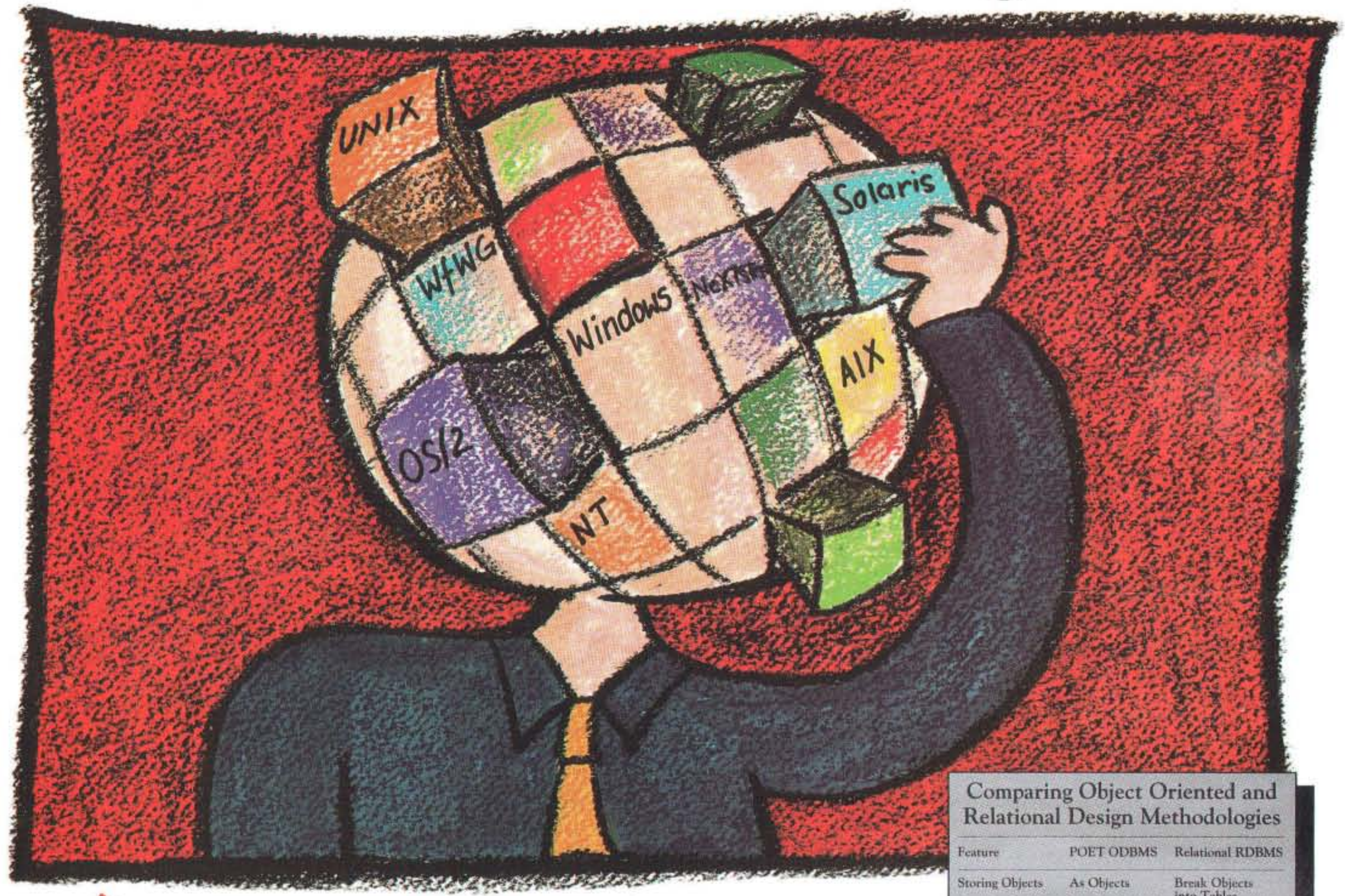
In the decryption case, Bob deciphers the cryptogram by solving for G; $G\equiv V_x(d_1,1)$ mod p. The multiplicative inverse of G can be calculated, modulo p, using the extended Euclidean algorithm (see Knuth), and the message is recovered by $M\equiv d_2(G^{-1})$ mod p. Figure 2 provides an example.

Note that the LUCELG cryptogram is twice the size it would be in LUC. Both $d_1$ and $d_2$ almost always have the same number of digits as the modulus, so the combined cryptogram will have a length of about twice that of p. This is also the case with the exponentiation version.

### Digital Signature

The El Gamal digital signature is more cumbersome to convert from exponentiation to Lucas functions than is El Gamal public-key encryption/decryption. However, observing that Lucas

# With our Object Database, All the Pieces Come Together.




**Plus**
**$299**
**Introductory Offer**

Smoothly. Easily. That's the beauty of POET, the Object-Oriented Database System for C++. It's simply a more efficient way to work, and yields finished applications that outperform relational applications in both productivity and performance.

What's the secret of Object Oriented Database programming? The POET ODBMS works with a one-step approach, seamlessly integrating into your application by storing your C++ objects in the database. By contrast, the relational approach forces you to design and maintain separate application and database models and then write lots of code to tie them together. So POET is more direct. More logical. And far more productive.

Compare POET with the relational design methodology you're using now. For a limited time, you can get the POET Personal Edition 2.1 for Windows for just $299*. **Or, call toll free to request a revealing white paper that compares the two methodologies and includes testimonials from key POET installed clients.** You'll find that POET is the solution to your application design puzzles.

**Comparing Object Oriented and Relational Design Methodologies**

| Feature | POET ODBMS | Relational RDBMS |
|---|---|---|
| Storing Objects | As Objects | Break Objects into Tables |
| Database Model | User Application Model | Separate Database Model Required |
| C++ Integration | Total | Poor |
| Database Operations | At Object Level | Must Write Code |
| Productivity | Increased | Reduced |
| Complex Object Performance | Excellent | Poor |

## Call 1-800-950-8845

**POET Software**

*This introductory offer is good through March 31, 1994. †POET applications are portable, at source and database level across all these popular platforms: Windows, OS/2, Macintosh, Windows NT, UNIX (Sun, Silicon Graphics, IBM), NEXTSTEP 486 and Novell (NLM)
POET Software Co. 4633 Old Ironsides Dr., Suite 110, Santa Clara, CA 95054 TEL: (408) 970-4640 FAX: (408) 970-4630

**CIRCLE NO. 877 ON READER SERVICE CARD**

om page 28)
e formulas for multiplying , subscripts — see Figure can construct an El Gamal-...r, since El Gamal's manipulation of exponents can be converted to the manipulation of Lucas-function subscripts. The formula for the addition of subscripts (Relation 2) involves the Lucas $\{U_i\}$ "sister" series. Subsequently, our Lucas-function alternative to El Gamal involves the doubling of the public-key size (two Lucas function values, U and V, must be given), as well as increasing the size of the signature, because two "r" (U and V) values are necessary.

Let the prime p be 908797. Choose k=1949, $\gamma$=19, x=2089, y=894501 and M=1111.

$G \equiv V_k(y,1) \bmod p = V_{1949}(894501,1) \bmod 908797 = 788038.$
$d_1 \equiv V_k(\gamma,1) \bmod p = V_{1949}(19,1) \bmod 908797 = 307718.$
$d_2 \equiv GM \bmod p = 788038.1111 \bmod 908797 = 338707.$

The cryptogram is the pair $(d_1,d_2)=(307718, 338707)$.
The receiver, who knows that the secret key is 2089, first calculates:

$G \equiv V_x(d_1,1) \bmod p = V_{2089}(307718,1) = 788038$ The inverse of this is 518288.
$M \equiv d_2(G^{-1}) = 338707.518288 = 1111$ The original message.

***Figure 2:*** *Example of LUCELG.*



The variant of El Gamal chosen as the Digital Signature Standard can be converted in a similar manner. In both cases, we produce ciphers apparently based on a problem for which there is no known subexponential-time attack; hence, they are stronger than their prototypes.

The calculation of the *n*th Lucas function can be done in O(log*n*) operations, which is the same order as the computation of similar exponentials. Heuristics to speed up modular exponentiation can be brought over to the calculation of Lucas functions, if in more complicated form (witness the formula for adding subscripts). These new ciphers can be assured of having performance characteristics similar to those of their progenitors.

### Conclusion

For the same level of security, these Lucas-function-based ciphers can be used with a shorter modulus than the exponentiation ciphers. For a 512-bit modulus, the reduction is about one fifth, down to 420 bits, for equivalent cryptographic strength. This reduction increases in size as the modulus grows longer. That only exponential-time attacks are possible on the Lucas-function version of the DL problem ensures attempts to solve it are increasingly more expensive than the subexponential-time attacks possible on the DL itself. We have applied for patents on these algorithms.

Finally, LUC Encryption Technology Ltd. (LUCENT), has been incorporated to license and support cryptographic systems based on Lucas functions. For more information, contact Horace R. Moore, 101 E. Bonita, Sierra Madre, CA 91024.

### References


Carey, M.R. and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: W.H. Freeman, 1979.

Diffie, W. and M.E. Hellman. "New Directions in Cryptography." *IEEE Transactions on Information Theory* (November 1976).

El Gamal, T. "A Public-key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." *IEEE Transactions on Information Theory* (July 1985).

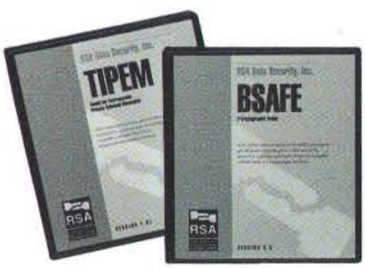
Knuth, D.E. *The Art of Computer Programming: Volume II: Semi-Numerical Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1981.

Smith, Peter. "LUC Public-Key Encryption." *Dr. Dobb's Journal* (January 1993).


**DDJ**

To vote for your favorite artile, circle inquiry no. 2.

**Security Clearance:**
The distance between you and the floor
after you learn how easy it is to use RSA.




**$290**

Developing applications with advanced encryption and authentication features doesn't have to be a headache. In fact, once you see how easy it is to work with the world's most trusted data security system, you might find yourself downright gleeful.

With BSAFE™ 2.0 or TIPEM™ 1.1, the latest releases of the world's best-selling crypto software developer's kits, you'll have everything you need to build secure applications. The one thing you won't need is a Ph.D. in number theory or cryptography. Both toolkits make using the patented RSA Public Key Cryptosystem (and a host of other algorithms) easy.

More importantly, your developers have the full support of RSA's staff of experienced mathematicians, cryptographers and engineers. You'll have the industry's leading cryptography shop standing behind your development effort–just like we stand behind other BSAFE™ and TIPEM™ customers like Apple, Microsoft, Lotus and Sun.

Call RSA at 415.595.8782. It could mean the difference between kicking up your heels or just kicking yourself.

RSA DATA SECURITY, INC.
THE KEYS TO PRIVACY AND AUTHENTICATION

**RSA. Because some things are better left unread.**

Object code versions of RSA toolkits are available for a variety of popular platforms, and source code licensing is also available. Contact RSA for details.



CIRCLE NO. 814 ON READER SERVICE CARD

# SHA: The Secure Hash Algorithm

## *Putting message digests to work*

William Stallings

An essential element of most authentication and digital-signature schemes is a hash algorithm. A hash function accepts a variable-size message M as input and produces a fixed-size tag H(M), sometimes called a "message digest," as output (see "One-Way Hash Functions," by Bruce Schneier, *DDJ*, September 1991). Typically, a hash code is generated for a message, encrypted, and sent with the message. The receiver computes a new hash code for the incoming message, decrypts the hash code that accompanies the message, and compares them. If the message has been altered in transit, there will be a mismatch.

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information-processing standard (FIPS PUB 180) in 1993. SHA is based on the MD4 algorithm, developed by Ron Rivest of MIT, and its design closely models MD4. SHA is used as part of the new Digital Signature Standard from NIST, but it can be used in any security application that requires a hash code.

*William is an independent consultant and president of Comp-Comm Consulting of Brewster, MA. This article is based on material in his forthcoming book,* Network and Internetwork Security *(Macmillan, due June 1994). He can be reached at stallings@acm.org.*

### SHA Logic

SHA takes as input a message with a maximum length of less than $2^{64}$ bits and produces as output a 160-bit message digest. The input is processed in 512-bit blocks. Figure 1 shows the overall processing of a message to produce a digest. The processing consists of the following steps:

**Step 1: Append padding bits.** The message is padded so that its length is congruent to 448 modulo 512. Padding


is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 512. The padding consists of a single 1 bit followed by the necessary number of 0 bits.

**Step 2: Append length.** A block of 64 bits is appended to the message. This block is treated as an unsigned 64-bit integer and contains the length of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. The figure represents the expanded message as the sequence of 512-bit blocks $Y_0, Y_1 \ldots Y_{L-1}$, so that the total length of the expanded message is L×512 bits. Equivalently, the result is a multiple of 16 32-bit words. Let M[0…N–1] denote the words of the resulting message, with N being an integer multiple of 16. Thus N=L×16.

**Step 3: Initialize MD buffer.** A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers (A,B,C,D,E). These registers are initialized to the following hexadecimal values (high-order octets first):

A=67452301
B=EFCDAB89
C=98BADCFE
D=10325476
E=C3D2E1F0

**Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a module that consists of 80 steps of processing; this module is labeled $H_{SHA}$ in Figure 1, and its logic is illustrated in Figure 2. The 80 steps have a similar structure.

Note that each round takes as input the current 512-bit block being processed ($Y_q$) and the 160-bit buffer value ABCDE, and updates the contents of the buffer. Each round also makes use of an additive constant $K_t$. Only four distinct constants are used. The values, in hexadecimal, are shown in Figure 3.

Overall, for block $Y_q$, the algorithm takes $Y_q$ and an intermediate digest value $MD_q$ as inputs. $MD_q$ is placed into buffer ABCDE. The output of the 80th step is added to $MD_q$ to produce $MD_{q+1}$. The addition is done independently for each of the five words in the buffer with each of the corresponding words in $MD_q$, using addition modulo $2^{32}$.

**Step 5: Output.** After all L 512-bit blocks have been processed, the output from the L*th* stage is the 160-bit message digest.

In the logic of each round, each round is of the form:

PARADOX 4.5 FOR WINDOWS DEVELOPMENT EDITION

# Accelerate into Client/Server with Paradox

If you're a Paradox® developer, you can easily create client/server applications from your existing code with the new Paradox 4.5 Development Edition for Windows. Now your applications can quickly gain server reliability, security, performance, and expandability.

### Paradox: Turn your applications into client/server solutions

Paradox Development Edition gives you all the power and security of Paradox, plus state-of-the-art object-oriented tools—reusable objects that let you quickly create and upsize applications.

You also get extensible tools for reengineering. So you can respond with new solutions to today's fast-changing technology and business trends. And with transparent Paradox-SQL connectivity, you can access data on SQL database servers just as if it were in local PC tables.

### InterBase: The most advanced RDBMS for upsizing

Paradox connects to Borland InterBase™ 3.3, which gives you advanced features that put you ahead of the competition. Paradox also connects easily to other popular database servers.


InterBase includes a powerful versioning engine to make your PC applications incredibly fast, even in a mixed read/write environment.

InterBase installs easily with a single command and tunes itself dynamically to the amount of disk and shared memory. Once it's installed, you can immediately use the Paradox SQL Tools included in your Development Edition to create and manage database tables with familiar Windows tools. It's an unbeatable combination.

### Borland: Training and support that get you up to speed

Whether you're in the evaluation stage—or building a full solution today—Borland makes it fast and easy to get on-the-spot technical support, with a full range of service options and annual contracts tailored to your special needs.

And with Borland's *automatic* software and documentation maintenance,* you're ensured product updates and upgrades for one year, the minute they're ready! Get Paradox today. It's the fastest way to client/server.

90-day, money-back guarantee!
**See your dealer or call now, 1-800-458-7730, ext. 8020**
In Canada call, 1-800-461-3327

**Borland**
Power made easy™



CIRCLE NO. 493 ON READER SERVICE CARD

*(continued from page 32)*

$$A,B,C,D,E \leftarrow [CLS_5(A)+f_t(B,C,D)+E+W_t+K_t],A,\ CLS_{30}(B),C,D$$

where A,B,C,D,E = the five words of the buffer; t = round, or step, number; $0 \leq t \leq 79$; $f_t$ = a primitive logical function; CLSs = circular left shift (rotation) of the 32-bit argument by s bits; $W_t$ = a 32-bit word derived from the current 512-bit input block; and $K_t$ = an additive constant. Four distinct values are used, and + = addition modulo $2^{32}$.

Each primitive function takes three 32-bit words as input and produces a 32-bit word output. Each function performs a set of bitwise-logical operations; that is, the n*th* bit of the output is a function of the n*th* bit of the three inputs. The functions are in Table 1. As you can see, only three different functions are used. For $0 \leq t \leq 19$, the function is the conditional function: *If B then C else D.* For $20 \leq t \leq 39$ and $60 \leq t \leq 79$, the function produces a parity bit. For $40 \leq t \leq 59$, the function is True if two or three of the arguments are True.

It remains to indicate how the 32-bit word values, $W_t$, are derived from the 512-bit message. The first 16 values of $W_t$ are taken directly from the 16 words of the current block. The remaining values are defined as:

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$

Thus, in the first 16 rounds of processing, the input from the message block consists of a single 32-bit word from that block. For the remaining 64 rounds, the input consists of the XOR of a number of the words from the message block.

SHA can be summarized as:

$$MD_0 = IV$$
$$MD_{q+1} = SUM_{32}(MD_q,\ ABCDE_q)$$
$$MD = MD_{L-1}$$

where IV = initial value of the ABCDE buffer, defined in Step 3; $ABCDE_q$ = the output of the last round of processing of the q*th* message block; L = the number of blocks in the message (including padding and length fields); $SUM_{32}$ = addition modulo $2^{32}$ performed separately on each word of the pair of inputs; and MD = final message-digest value.

*SHA is used as part of the new Digital Signature Standard from NIST but can be used in any security application that requires a hash code*

## SHA Security

SHA has the property that every bit of the hash code is a function of every bit in the input. The complex repetition of the basic function $f_t$ produces well-mixed results; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Unless there is some hidden weakness in SHA, which has not so far been published, the difficulty of coming up with two messages having the same message digest is on the order of $2^{80}$ operations, while the difficulty of finding a message with a given digest is on the order of $2^{160}$ operations.

**DDJ**

To vote for your favorite article, circle inquiry no. 3.

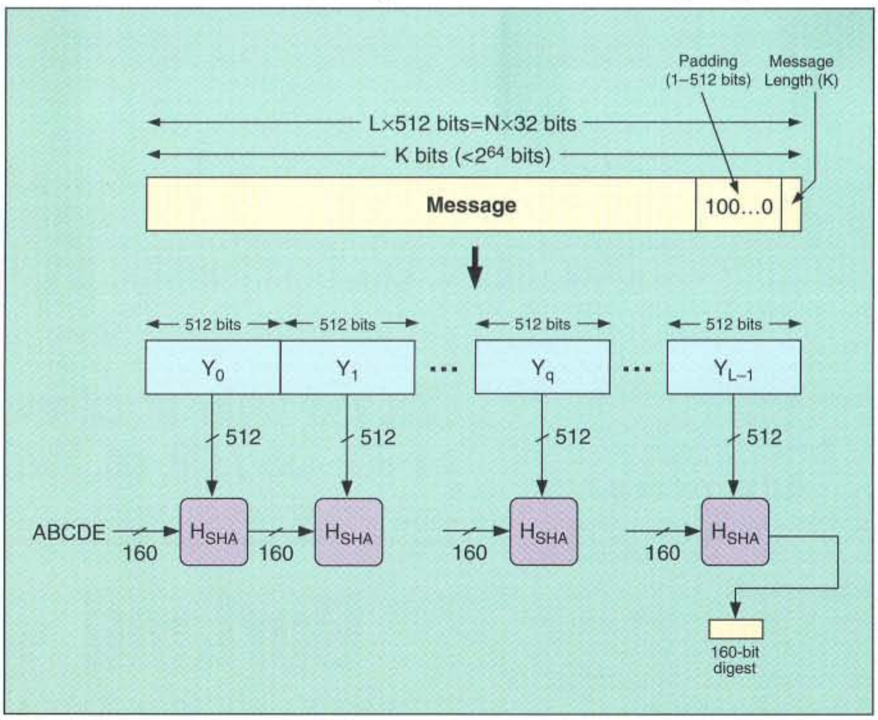



**Figure 1:** *Overall processing of a message to produce a digest.*

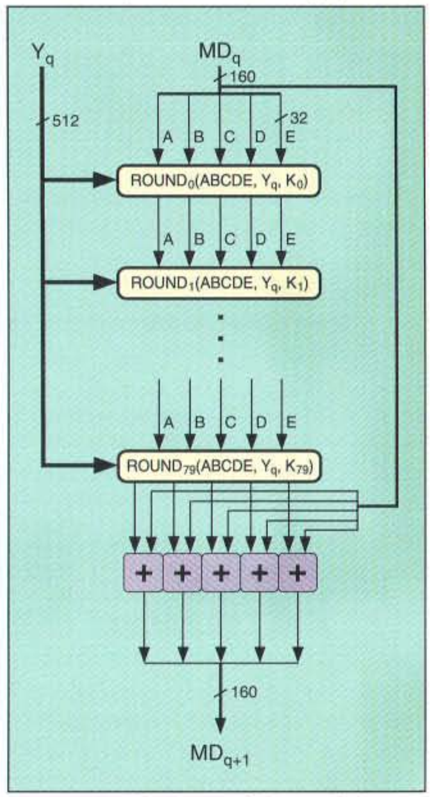


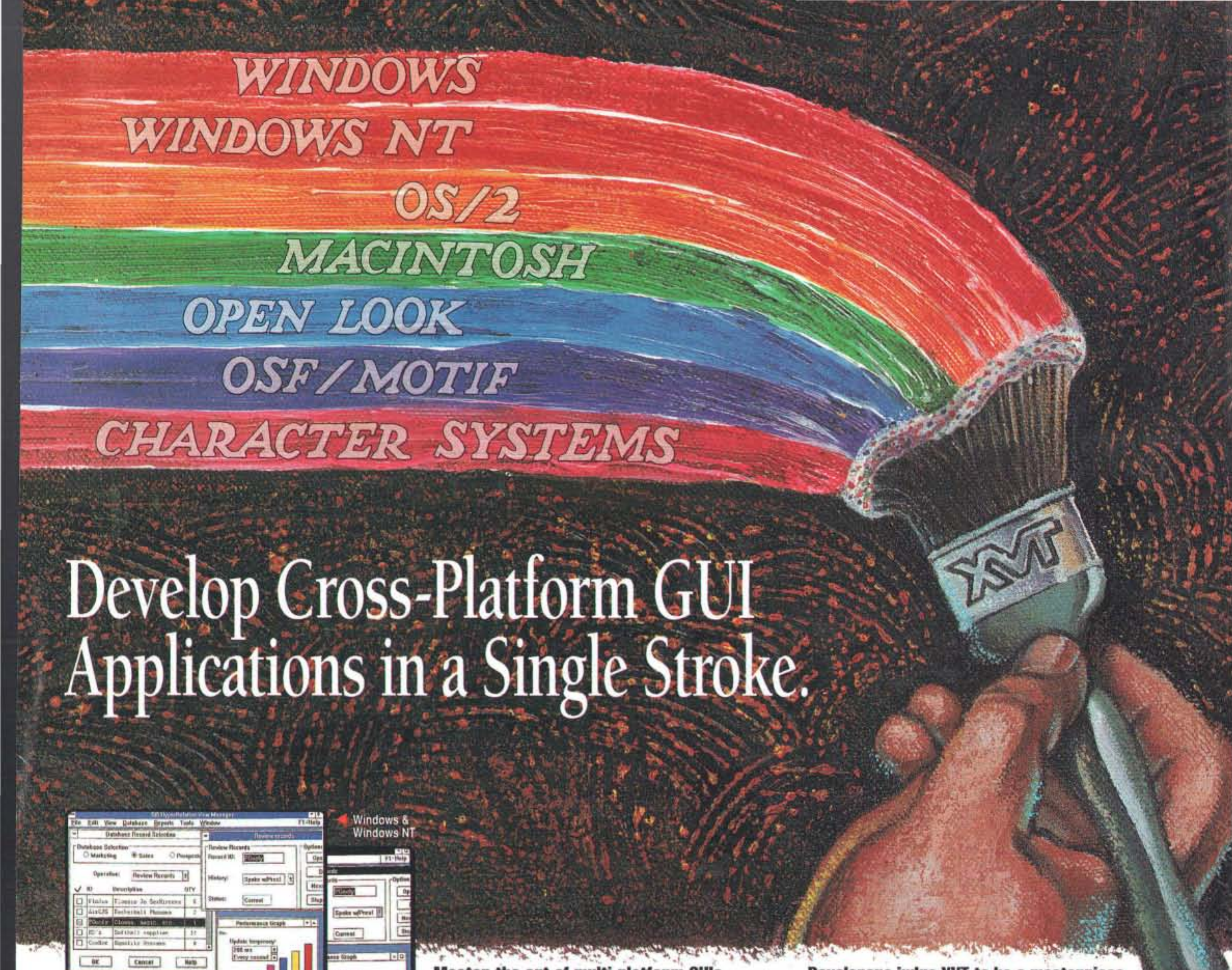

**Figure 2:** *The logic of the module $H_{SHA}$; addition (+) is mod $2^{32}$.*

| | |
|---|---|
| $0 \leq t \leq 19$ | $K_t$=5A827999 |
| $20 \leq t \leq 39$ | $K_t$=6ED9EBA1 |
| $40 \leq t \leq 59$ | $K_t$=8F1BBCDC |
| $60 \leq t \leq 79$ | $K_t$=CA62C1D6 |

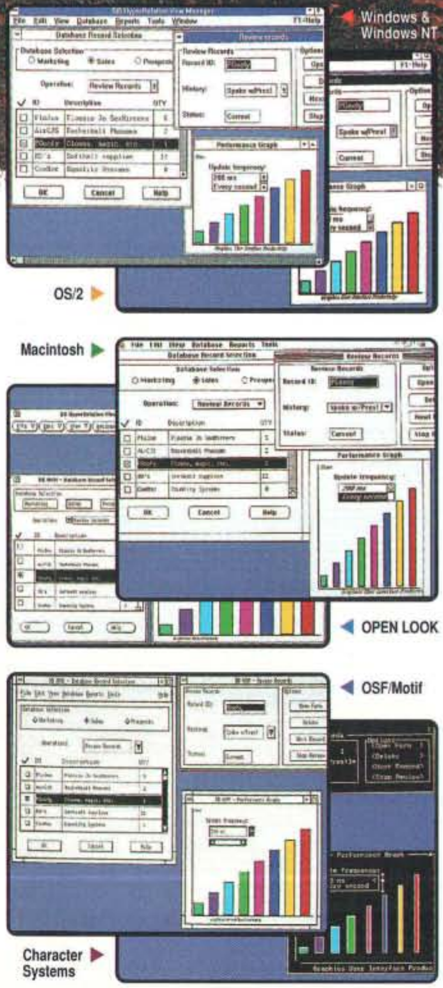
**Figure 3:** *Hexadecimal values of the four constants.*

| Round | $f_t(B,C,D)$ |
|---|---|
| $(0 \leq t \leq 19)$ | $(B \cdot C) \vee (\overline{B} \cdot D)$ |
| $(20 \leq t \leq 39)$ | $B \oplus C \oplus D$ |
| $(40 \leq t \leq 59)$ | $(B \cdot C) \vee (B \cdot D) \vee (C \cdot D)$ |
| $(60 \leq t \leq 79)$ | $B \oplus C \oplus D$ |

**Table 1:** *SHA primitive functions.*




# Develop Cross-Platform GUI Applications in a Single Stroke.




**Master the art of multi-platform GUIs.**

XVT Software is the leading choice of world-class developers for one reason: It is the simplest, quickest path to building quality applications that port to every GUI without compromises in look-and-feel or performance. Plus, it's easier to learn and use than native toolkits, so your time and effort goes into your application, not your GUIs.

**XVT gives you simultaneous original GUIs.**

Because XVT uses native GUI objects, your application is indistinguishable from one written directly to the native toolkit. Through our layered architecture, you achieve equivalent cross-platform functionality appropriate to each GUI, without the overhead and inflexibility of proprietary emulation-based systems.

**XVT puts complete C/C++ solutions at your fingertips.**

XVT Development Solutions for C include an Interactive Design Tool. Solutions for C++ include an object-oriented application framework. Both include the XVT Portability Toolkit.

When combined with in-depth consulting, training and support, plus a wide range of Partners products, XVT forms the most comprehensive and advanced solution for developing completely portable GUI applications.

NEW! XVT-PowerObjects

**Developers judge XVT to be a masterpiece.**

XVT is the base document for the IEEE's GUI standardization effort. Our thousands of customers include internal and commercial developers like: Alcoa, Amoco, AT&T, Avis, Ford, General Motors, Grammatik/Reference Software, Kodak, Lockheed, NCR, NEC, NIST, Novell, Rockwell, Siemens, Sony, Southwestern Bell, Tandem, Uniplex, Unisys, US Army and US West.

**Call now for a Free XVT Demo and Technical Overview.**

**1-800-678-7988**

XVT SOFTWARE INC

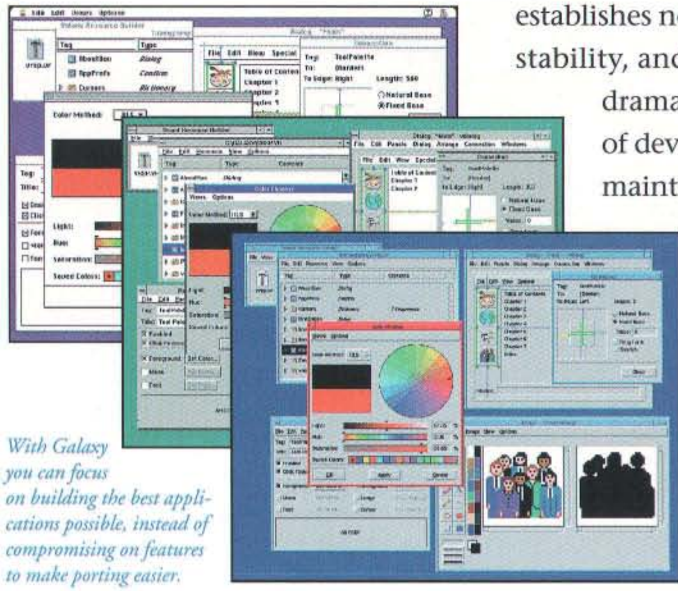
The portable GUI development solution.

XVT Software Inc. 4900 Pearl East Cir. Boulder, CO 80301
(303) 443-4223 FAX (303) 443-0969

For European inquiries, contact: Precision Software GmbH
Phone: 49 0 61 03/37 94 0 Fax: 49 0 61 03/36 95 5

**CIRCLE NO. 655 ON READER SERVICE CARD**

If you're building a new generation of mission-critical applications, you need a tool with the power and scalability to span departments, divisions—the entire enterprise. Only one tool can help you do it all, because only one tool has it all—the Galaxy Application Environment® from Visix.



*With Galaxy you can focus on building the best applications possible, instead of compromising on features to make porting easier.*

establishes new standards for power, stability, and completeness—it dramatically reduces the cost of developing, enhancing, and maintaining high-performance distributed applications. Your *entire* Galaxy application can easily be moved to UNIX, Macintosh, Windows, Windows NT, OS/2, and OpenVMS without changing a single line of code.

# IT'S ALL IN THE

## A New Architectural Standard

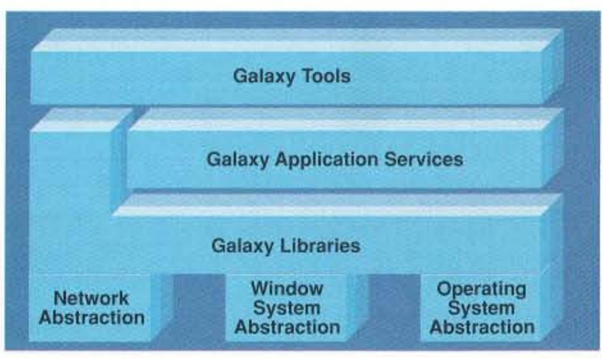
If you want to build it once and build it right, then choose your development tool based on what's built-in from the start, not what's bolted-on later. Galaxy is the only application development environment designed specifically for building the next generation of large-scale corporate applications.

"An unbelievable product!"
**—*UNIX Review***


The Galaxy architecture frees you to concentrate on building portable best-of-breed applications, instead of mastering endless operating system, window system, and networking details for each platform. The fully object-oriented Galaxy API

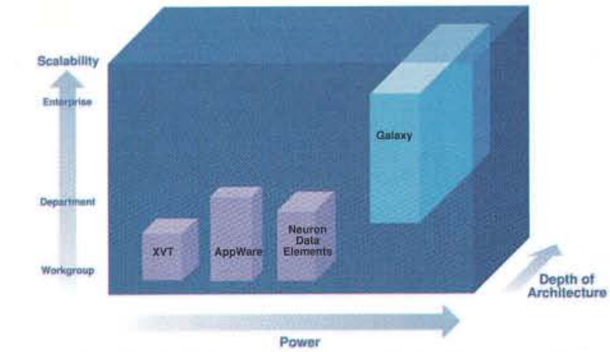
## A Complete Solution

In a marketplace crowded with partial solutions such as GUI builders and class libraries, Galaxy stands apart from the pack by delivering a complete multiplatform development environment. Galaxy specifically targets the needs of high-end professional developers by assembling a comprehensive set of innovative tools, libraries, and runtime services into one cohesive development




*Galaxy's object-oriented architecture offers unprecedented API depth and breadth to liberate C and C++ developers from the constraints of platform-specific toolkits.*

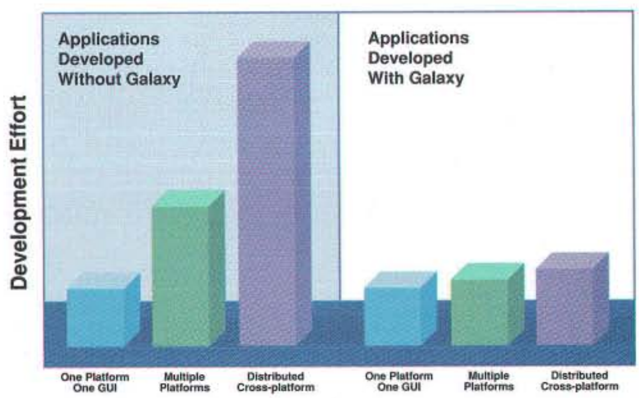
VISIX




*Only Galaxy offers the high-end solution you need to build mission-critical applications that scale across departments, divisions—the global enterprise.*

environment. Of course, Galaxy delivers a world-class GUI builder. But, Galaxy goes far beyond simple GUI builders to deliver a rich set of platform-independent abstractions—

# ARCHITECTURE




*The powerful Galaxy architecture enables you to bring your mission-critical applications in on-time and under-budget.*

file management, networking, memory management, and distributed services—to handle the *other* 50% of your project. For example, the built-in Galaxy Distributed Application Services™ component provides you with the most advanced network development environment available, supporting both heterogeneous client-server and peer-to-peer architectures.

## Our Pain is Your Gain

In building Galaxy, we learned first-hand that "bolt-ons" don't work. Over the past seven years, we've poured hundreds of engineer-years into designing and re-designing Galaxy to ensure that your Galaxy applications will be limited only by your imagination. The result: Galaxy now stands at over 1.5 million lines of efficient, stable code. Yet, Galaxy programs are typically smaller and faster than those built using native toolkits. Unbelievable? Ask the editors of *UNIX Review*, or better yet, ask Galaxy customers. Galaxy's architecture has proven itself in hundreds of real world environments.

## Field-proven

From Wall Street to Silicon Valley, Galaxy has exceeded the expectations of even the most demanding developers *and* won the praises of computer industry editors and analysts. Hundreds of leading corporations and major ISVs have adopted Galaxy—J.P. Morgan, Unify, IBM, Hewlett-Packard, LEGENT, Xerox, and many others are using Galaxy *today* to build their mission-critical applications.

## Get the Kit


See for yourself. We'll send you everything you need to know to begin exploring Galaxy, including industry analysis, seminar information, and an in-depth technical overview. Call us at 800.832.8668 or 703.758.8230 today. Or, if you prefer, send your request by fax to 703.758.0233 or via email to galaxy@visix.com.

**GALAXY**
Application Environment®
LIGHT YEARS AHEAD

**Visix Software Inc.** **800.832.8668/toll free** **703.758.8230/phone** **703.758.0233/facsimile** **galaxy@visix.com/email**

CIRCLE NO. 754 ON READER SERVICE CARD

# The Blowfish Encryption Algorithm

## *A fast, new algorithm for 32-bit CPUs*

Bruce Schneier

Blowfish is a block-encryption algorithm designed to be fast (it encrypts data on large 32-bit microprocessors at a rate of 26 clock cycles per byte), compact (it can run in less than 5K of memory), simple (the only operations it uses are addition, XOR, and table lookup on 32-bit operands), secure (Blowfish's key length is variable and can be as long as 448 bits), and robust (unlike DES, Blowfish's security is not diminished by simple programming errors).

The Blowfish block-cipher algorithm, which encrypts data one 64-bit block at a time, is divided into key-expansion and a data-encryption parts. Key expansion converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes. Data encryption consists of a simple function iterated 16 times. Each iteration, called a "round," consists of a key-dependent permutation and a key- and data-dependent substitution.

### Subkeys

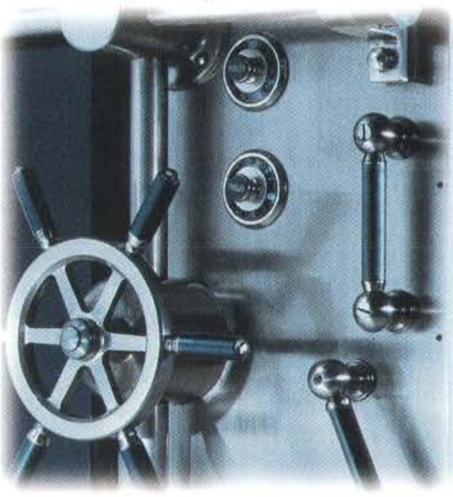
Blowfish uses a large number of subkeys that must be precomputed before any data encryption or decryption. The P-array consists of 18 32-bit subkeys, $P_1, P_2 \ldots P_{18}$, and there are four 32-bit S-boxes with 256 entries each: $S_{1,0}, S_{1,1} \ldots S_{1,255}$; $S_{2,0}, S_{2,1} \ldots S_{2,255}$; $S_{3,0}, S_{3,1} \ldots S_{3,255}$; $S_{4,0}, S_{4,1} \ldots S_{4,255}$.

*Bruce is the author of* Applied Cryptography: Protocols, Algorithms, and Source Code in C *(John Wiley, 1994). This article is based on a paper he presented at the Cambridge Algorithms Conference. Bruce can be contacted at schneier@chinet.com.*

### Encryption

Blowfish is a Feistel network consisting of 16 rounds; see Figure 1. The input is a 64-bit data element, x. Divide x into two 32-bit halves: $x_L$ and $x_R$. Then, for i=1 to 16:

$x_L = x_L \text{ XOR } P_i$
$x_R = F(x_L) \text{ XOR } x_R$
Swap $x_L$ and $x_R$



After the sixteenth iteration, swap $x_L$ and $x_R$ to undo the last swap. Then $x_R = x_R \text{ XOR } P_{17}$ and $x_L = x_L \text{ XOR } P_{18}$. Finally, recombine $x_L$ and $x_R$ to get the ciphertext.

Function F looks like this: Divide $x_L$ into four eight-bit quarters: a, b, c, and d. $F(x_L)=((S_{1,a}+S_{2,b} \bmod 2^{32}) \text{XOR } S_{3,c})+ S_{4,d} \bmod 2^{32}$; see Figure 2.

Decryption is exactly the same as encryption, except that $P_1, P_2 \ldots P_{18}$ are used in the reverse order.

Implementations of Blowfish that require the fastest speeds should unroll the loop and ensure that all subkeys are stored in cache. For the purposes of illustration, I've implemented Blowfish in C; Listing One (page 98) is blowfish.h, and Listing Two (page 98) is blowfish.c. A required data file is available electronically; see "Availability," page 3.

### Generating the Subkeys

The subkeys are calculated using the Blowfish algorithm, as follows:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed random string. This string consists of the hexadecimal digits of $\pi$.
2. XOR $P_1$ with the first 32 bits of the key, XOR $P_2$ with the second 32 bits of the key, and so on for all bits of the key (up to $P_{18}$). Cycle through the key bits repeatedly until the entire P-array has been XORed.
3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps #1 and #2.
4. Replace $P_1$ and $P_2$ with the output of step #3.
5. Encrypt the all-zero string using the Blowfish algorithm with the modified subkeys.
6. Replace $P_3$ and $P_4$ with the output of step #4.
7. Continue the process, replacing all elements of the P-array and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than re-executing this derivation process.

### Design Decisions

The underlying philosophy behind Blowfish is that simplicity of design yields an algorithm that is easier both to understand and to implement. Hopefully, the use of a streamlined Feistel network (the same structure used in DES, IDEA, and many other algorithms), a simple S-box substitution, and a simple P-box substitution, will minimize design flaws.

# Write It Once.

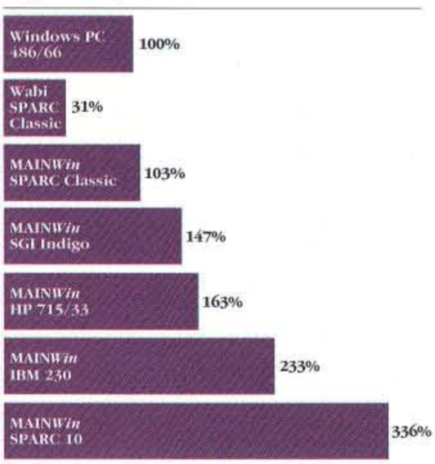

*Windows*

# Sell It Everywhere.


*SPARC Classic* *HP715/33* *IBM 230* *SGI Indigo*

Interested in increasing your potential market by 20%, or more? Then take a close look at MainWin technology.

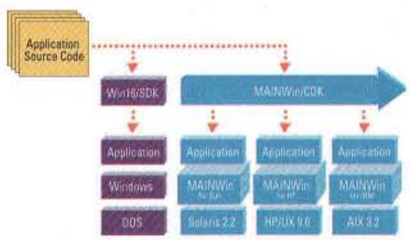
***% of 486 Performance****




** Composit Code and Library benchmarks. Contact MainSoft for full details.*

Far beyond the usual porting strategies, MainWin technology gives you a completely new and revolutionary cross-development paradigm. Built around a robust Windows API compatible library for UNIX, MainWin lets you leverage the Windows code base you have now into the UNIX workstation market. Not only that, in the future, MainWin will save you time, trouble and money by letting you create software products for both markets from a single code base.

And talk about performance. MainWin technology actually recompiles your Windows 3.1 code (written in C or C++ — including MFC support) to run as native code in whatever environment you've targeted. So your new UNIX applications take full advantage of their new RISC-based environments.




To find out more, call MainSoft at **1-800-MAINWIN.** And see how MainWin technology can open up profitable opportunities for you everywhere.

**MAIN Soft**

883 N. Shoreline Blvd.
Suite C-100
Mountain View, CA 94043

*(continued from page 38)*

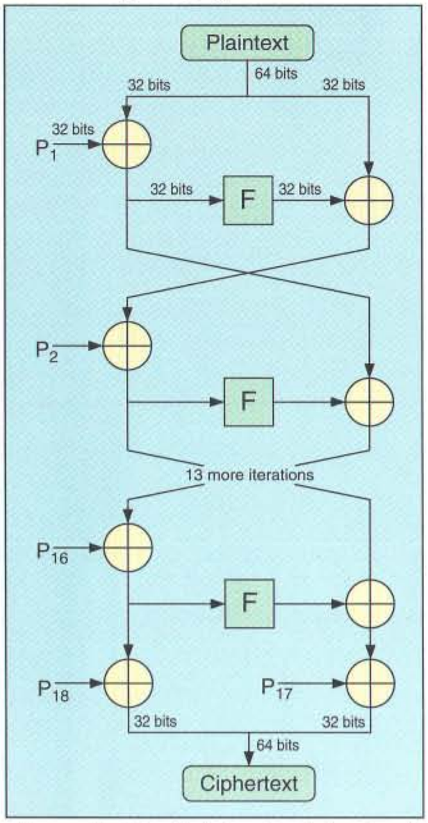



***Figure 1:** Blowfish is a Feistel network consisting of 16 rounds.*

For details about the design decisions affecting the security of Blowfish, see "Requirements for a New Encryption Algorithm" (by B. Schneier and N. Fergusen) and "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)" (by B. Schneier), both to be included in *Fast Software Encryption,* to be published by Springer-Verlag later this year as part of their *Lecture Notes on Computer Science* series. The algorithm is designed to be very fast on 32-bit microprocessors. Operations are all based on a 32-bit word and are one-instruction XORs, ADDs, and MOVs. There are no branches (assuming you unravel the main loop). The subkey arrays and the instructions can fit in the on-chip caches of both the Pentium and the PowerPC. Furthermore, the algorithm is designed to be resistant to poor implementation and programmer errors.

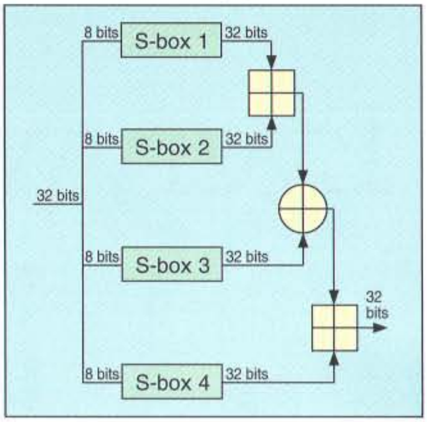



***Figure 2:** Blowfish function F.*

I'm considering several simplifications to the algorithm, including fewer and smaller S-boxes, fewer rounds, and on-the-fly subkey calculation.

### Conclusions

At this early stage, I don't recommend implementing Blowfish in security systems. More analysis is needed. I conjecture that the most efficient way to break Blowfish is through exhaustive search of the keyspace. I encourage all cryptanalytic attacks, modifications, and improvements to the algorithm.

However, remember one of the basic rules of cryptography: The inventor of an algorithm is the worst person to judge its security. I am publishing the details of Blowfish so that others may have a chance to analyze it.

Blowfish is unpatented and will remain so in all countries. The algorithm is hereby placed in the public domain and can be freely used by anyone.

**DDJ**

**(Listings begin on page 98.)**

To vote for your favorite article, circle inquiry no. 4.

# *DDJ's* Blowfish Cryptanalysis Contest

The only way to inspire confidence in a cryptographic algorithm is to let people analyze it. It is in this spirit that *DDJ* is pleased to announce the Blowfish Cryptanalysis Contest, our third reader contest in recent years.

We'd like you to cryptanalyze Bruce Schneier's Blowfish algorithm presented in this issue. Give it your best shot. Break it, beat on it, cryptanalyze it. The best attack received by April 1, 1995 wins the contest.

The contest rules are simple. It's open to any individual or organization. Governments are encouraged to enter. Even the NSA can compete and win the prize (their budget isn't what it used to be; they can probably use the money). But since we will publish the results, classified entries will not be permitted. To officially enter the contest, your entry must be accompanied by a completed and signed entry form. These are available electronically (see "Availability," page 3) or we'll be glad to mail or fax you a hardcopy.

We're not going to publish messages encrypted in Blowfish and some random key, because we think that would be too difficult.

Partial results— those attacks that don't break the algorithm but instead prove that it isn't as strong as we thought it was— are just as useful and can be entered.

Your entry does not have to consist of code. Instead, your entry can be a paper describing the attack. The attack does not have to completely break the Blowfish algorithm, it can simply be more efficient than a brute-force attack. The attack can be against either the complete algorithm or a simplified version of the algorithm (fewer rounds, smaller block size, simpler S-boxes, and the like).

We'll select a winner based on the following criteria:

- Success of the attack. How much more efficient is the attack than brute force?
- Type of attack. Is it ciphertext only, known plaintext, or chosen plaintext?
- Type of algorithm. Is the attack against full Blowfish or a simplified version of the algorithm?

Bruce Schneier, frequent *DDJ* contributor, author of *Applied Cryptography,* and inventor of the Blowfish algorithm will referee the contest.

The contest results will be published in the September 1995 issue of *Dr. Dobb's Journal,* in which we'll discuss and summarize the winning programs, the weaknesses of the Blowfish algorithm, and any modifications of the algorithm.

We'll be providing a number of awards for the winners. The grand-prize winner will receive a $750 honorarium. Honorariums of $250 to the second-place winner and $100 to the third-place winner will also be awarded.

—editors

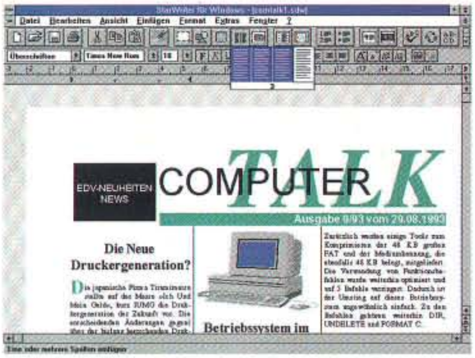
# WINDOWS

# MAC


**For more information on the only C++ application framework that you need to know about for porting your**


# UNIX

# OS/2




# YOU ONLY HAVE TO WRITE IT ONCE WITH STARVIEW 2.0

By learning StarView,™ you can write high-end applications with a single source code for all of our supported platforms with their native look and feel and only have to debug them once—cutting your development time dramatically. StarView can guarantee seamless portability because it was designed with a multi-platform approach from the very beginning, and it has always supported the latest features—necessary for competitive high-end applications.

**StarView 2.0 Features**

StarView has all the standard controls like Windows, Dialogs, Menus, and Buttons. But it also has other portable capabilities like Help System, Drag & Drop, OLE, Bitmaps, Toolbar, Statusline, Tablecontrol, MDI, Printing and Preview, Internationalizing of Date, Time and Metric Classes, and a Multinational Resource System that allows you to create a single Resource file containing all international versions of your program. StarView also has standard dialogs for File-Handling, Printer-Setup, and Color- and Font-Selection.

All controls are also available in the 3D StarView Look. And in addition to the program features, StarView comes standard with the StarView Design Editor and the StarView SFX Application Framework.

**applications to Windows, Windows NT, Macintosh, UNIX, and OS/2 environments, call 1-800-888-8527.**

**Thousands of Programmers Prefer StarView**

We've developed the most complete and clearly implemented C++ multi-platform software development tools available. That's why, even after trying other libraries, thousands of programmers still prefer using StarView—from the smallest startup to the largest computer communications and systems integration companies in the U.S.

**Call 800-888-8527 for More Information**

If you're interested, call us to receive a full working democopy of StarView, with DesignEd, and the StarView Whitepaper. You'll be able to explore the features of StarView as well as discover how easy it is to use.

**Pricelist**

| | |
|---|---|
| Windows 3.1 | $498 |
| Apple Macintosh | $498 |
| IBM OS/2 2.1 | $498 |
| Windows NT/Intel | $498 |
| Windows NT/Dec Alpha | $698 |
| Sun SPARC Solaris 2.x/Motif | $1498 |
| IBM RS 6000/Motif | $1498 |
| DEC Alpha OSF/1 | $1498 |

Call for information on other platforms.

**Star Division Corporation**
2180 Sand Hill Road, Suite 320
Menlo Park, CA 94025
Tel: 415.233.0140
Fax: 415.233.0142
E-mail: svinfo@stardiv.de

STARDIVISION™

CIRCLE NO. 782 ON READER SERVICE CARD

# The Wavelet Packet Transform

## *Extending the wavelet transform*

Mac A. Cody

The wavelet transform enables analysis of data at multiple levels of resolution (also known as "scale"). In addition, transient events in the data are preserved by the analysis. When the wavelet transform (WT) is applied to a signal in the time domain, the result is a two-dimensional, time-scale domain analysis of the signal. The transform has proven useful for the compression and analysis of signals and images.

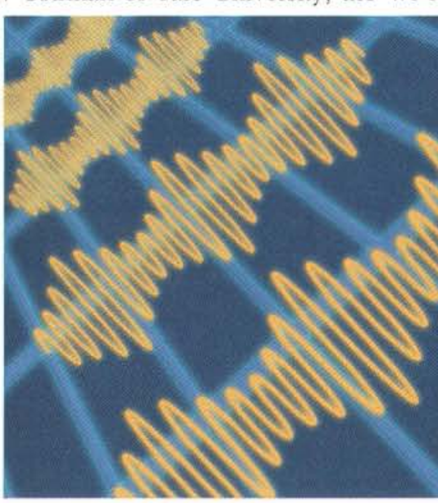
The fast wavelet transform (FWT) is an efficient implementation of the discrete wavelet transform (DWT). The DWT is the WT as applied to a regularly sampled data sequence. The transform of the data exhibits discrete steps in time on one axis, and discrete steps of resolution on another. The algorithm and a C language implementation of the FWT were presented in my article, "The Fast Wavelet Transform" (*DDJ,* April 1992).

As demonstrated in that article, the superiority of the DWT over the discrete Fourier transform (DFT) is in the DWT's simultaneous localization of frequency and time, something that DFTs can't do. As a trade-off, the frequency divisions in the DWT are not in integral steps. Instead, the divisions are in octave bands. Each level of the transform represents a frequency range half as wide as that of the level above it and twice as wide as that of the level below it; see Figure 1(a).

*Mac is an engineering specialist at E-Systems' Garland Division in Dallas, Texas. He can be contacted at 214-205-6452 and on Internet at mcody@aol.com.*

Conversely, the time scale on each level is twice that of the level below it and half that of the level above it; see Figure 1(b). This characteristic of the DWT poses problems when attempting to localize higher frequencies. Discrimination of frequency is sacrificed for time localization at the higher levels in the transform.

It turns out that the DWT is actually a subset of a far more versatile transform, the wavelet packet transform (WPT). Developed by Dr. Ronald A. Coifman of Yale University, the WPT



generalizes the time-frequency analysis of the wavelet transform. It yields a family of orthonormal transform bases of which the wavelet transform basis is but one member.

In this article I'll develop the wavelet packet transform algorithm from its roots in the wavelet transform algorithm. After that, I'll present C code to implement the algorithm.

### From a Humble Root a Tree Shall Grow

In the fast wavelet transform algorithm, the sampled data set is passed through the scaling and wavelet filters (the convolution operation). They are, respectively, low-pass and high-pass filters with complementary bandwidths, also known as a quadriture mirror filter (QMF) pair. The outputs of both filters are decimated (desampled) by a factor of two. The high-pass filtered data set is the wavelet transform detail coefficients at that level of scale of the transform. The low-pass filtered data set is the approximation coefficients at that level of scale. Due to the decimation, both sets of coefficients have half as many elements as the original data set.

The approximation coefficients can now be used as the sampled data input for another pair of wavelet filters, identical to the first pair, generating another set of detail and approximation coefficients at the next-lower level of scale. This process can continue until the limit for the unit interval is reached. For example, if it is desired that the transform have six levels (5 through 0), then the unit interval must be 64 ($2^6$) samples long. The data set can be of any length as long as it has an integral number of unit intervals. The resulting algorithm is the forward fast wavelet transform tree algorithm; see Figure 2(a).

You can turn the tree algorithm on end, with the initial data input at the top and the detail and approximation coefficients fanning out towards the bottom. The fast wavelet transform algorithm can now be viewed as a partial graph of a binary tree (the significance of this will be seen shortly); see Figure 2(b). The flow of the algorithm moves down and to the left, forming new levels of the transform from the approximation coefficients at higher levels. The detail "branches" are not used for further calculations.

Observe that the wavelet transform operation can be stopped at any level while working down the tree. The resulting "partial" transform is still a valid orthonormal transform. For example, if the unit interval for a data set were 32 points ($2^5$), the corresponding transform would have five levels (4 through 0). If the transform operation were

stopped at level 2, the transform would have only three levels, but the approximation and detail coefficients of the transform would correspond exactly to a wavelet transform with a unit interval of 8 ($2^3$) samples; see Figure 2(b).

The implication of this observation is that the QMF pair is an orthonormal transform kernel, just as butterfly operation is the kernel of the FFT. As long as the filters are designed to be orthonormal wavelet filters and the original data set meets the unit-interval requirement described above, repeated applications of the kernel will always yield an orthonormal transform.

Now, the set of detail and approximation coefficients at each level of the transform forms a pair of subspaces of the approximation coefficients of the next-higher level of scale and, ultimately, of the original data set. The subspaces created by the wavelet transform roughly correspond to the frequency subbands shown in Figure 1(a). These subspaces form a disjoint cover of the frequency space of the original data set. In other words, the subspaces have no elements in common, and the union of the frequency subbands span the frequency range of the original data set.

What Coifman proposed is that any set of subspaces which are a disjoint cover of the original data set is an orthonormal basis. The wavelet transform basis is then but one of a family of orthonormal bases with different subband intervals. As with the wavelet transform basis, each disjoint cover roughly corresponds to a covering of the frequency space of the original signal. Coifman dubbed this family a "wavelet packet library." The various orthonormal bases are formed by arbitrary applications of the orthonormal transform kernel upon the detail coefficients as well as the approximation coefficients of higher transform levels.

The application of the transform kernel to both the detail and approximation coefficients results in an expansion of the structure of the fast wavelet transform tree algorithm. The tree algorithm for the wavelet packet transform can be represented as a full binary tree; see Figure 3. As read from left to right, the *a* and *d* symbols at each node indicate the order of orthonormal transform kernel filter operations performed which yield each particular subspace of the original data set. Each node in the transform tree is also representative of a particular wavelet packet. The transform coefficients computed at each node are a correlation of the original data set and a waveform function representing the wavelet packet.

For example, the sequence *aaad* in Figure 3 represents four operations of the orthonormal transform kernel representing one of 48 possible wavelet packets. (The combination of all possible translations in time and dilation in scale for the wavelet packets is $J*2^J$; in this instance J equals 4.) The first three represent low-pass filter/decimation operations performed by the transform kernel. The fourth represents a high-pass filter/decimation operation performed by the transform kernel. This subspace should be recognizable as exactly the level 0 detail coefficients of the wavelet transform. The operations of the orthonormal transform kernel correspond to the wavelet function of the wavelet transform. Likewise, the wavelet packet represented by *aaaa* is the scaling function of the wavelet transform.

### Packets, Graphs, and Bases

The wavelet transform basis is actually a subset of a family of bases formed by the wavelet packet transform. The heavy lines in Figure 3 indicate the graph forming the wavelet basis. Note that the wavelet basis consists of the subspaces

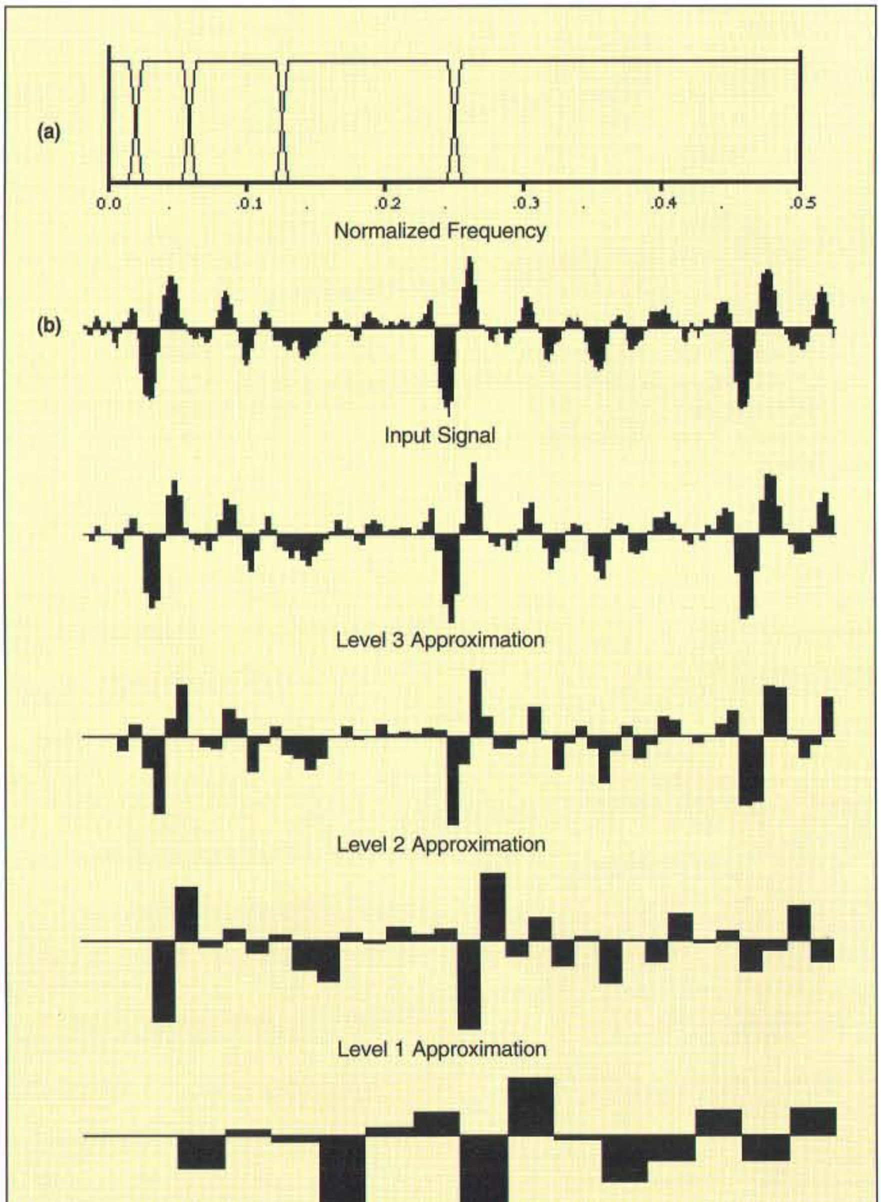



***Figure 1:*** *(a) The Discrete Wavelet Transform (DWT) divides the spectrum of the sampled data into octave bands; (b) the resolution at each level of the DWT is half that of the level above it and twice that of the level below. At lower levels, time resolution is sacrificed for frequency localization.*

*d*, *ad*, *aad*, *aaad*, and *aaaa*. The sequences *a*, *aa*, and *aaa* are intermediate steps leading to the generation of the subspaces of the wavelet basis at the lower levels. Since the orthonormal transform kernel can be arbitrarily applied to either approximation or detail branches on the tree, $J*2^J$ graphs representing different orthonormal bases can be created; see Figure 4.

The variety of orthonormal bases which can be formed by the WPT algorithm, coupled with the infinite number of wavelet and scaling functions which can be created, yields a very flex-

*(continued on page 50)*

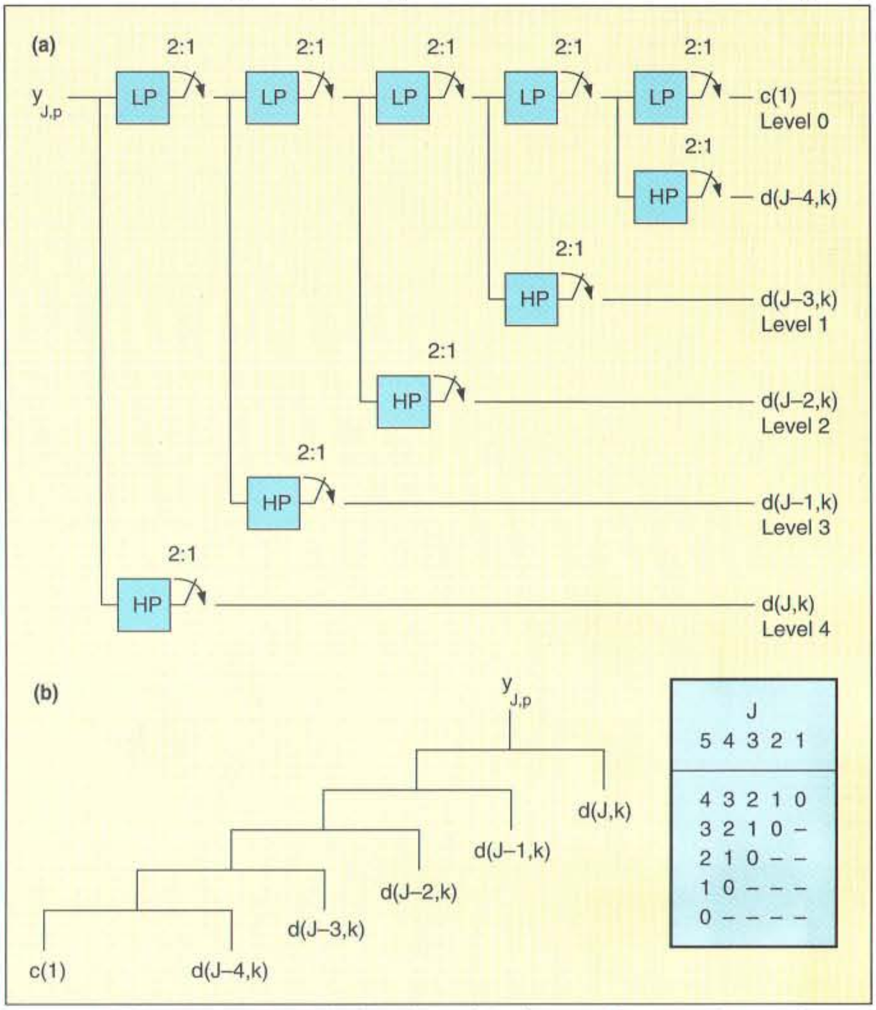



***Figure 2:*** *The tree or pyramid algorithm of the forward fast wavelet transform (a) can be viewed as a partial graph of a binary tree (b). For a particular unit interval ($2^J$ samples), a maximum of J levels of transform data can be formed.*

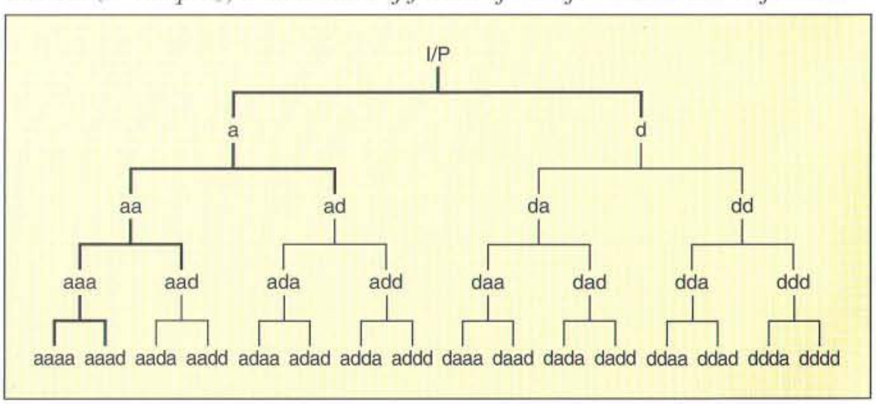
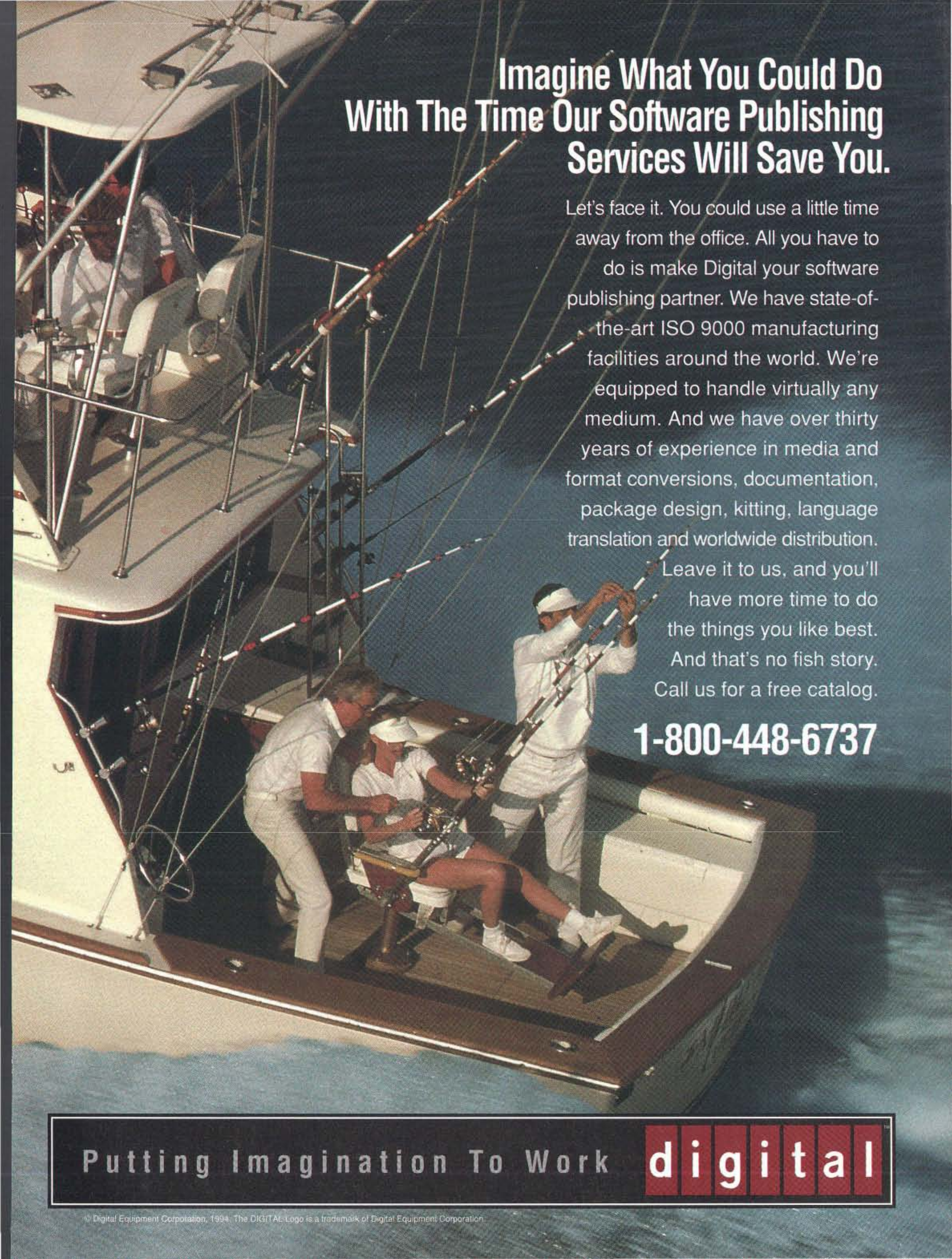
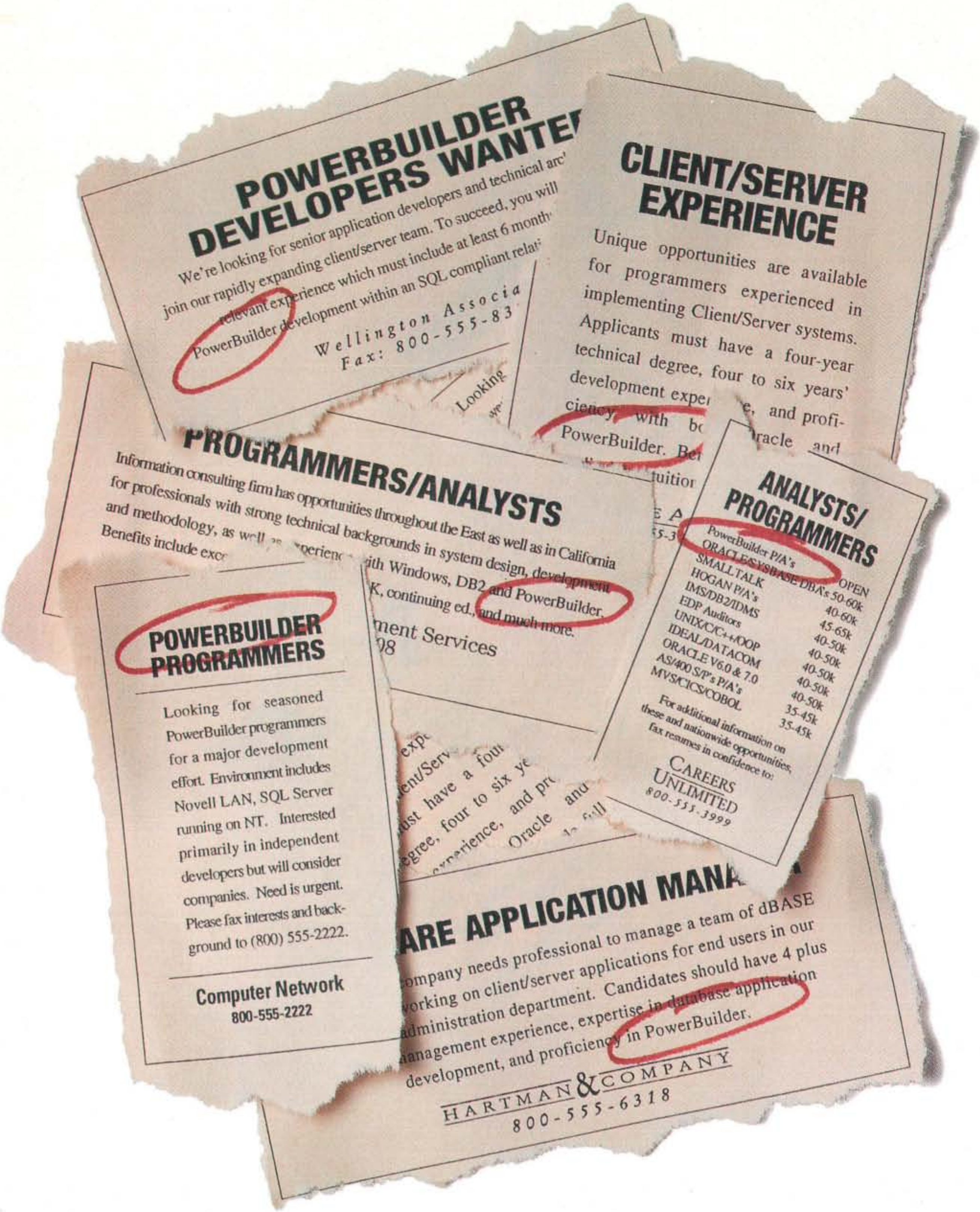



***Figure 3:*** *The wavelet packet transform viewed as a complete binary graph. Each "a" and "d" in each sequence represents the filtering operations performed to yield the particular subspace of the original signal. The bold lines represent the disjoint cover known as the "wavelet basis."*



Imagine What You Could Do With The Time Our Software Publishing Services Will Save You.
Let's face it. You could use a little time away from the office. All you have to do is make Digital your software publishing partner. We have state-of-the-art ISO 9000 manufacturing facilities around the world. We're equipped to handle virtually any medium. And we have over thirty years of experience in media and format conversions, documentation, package design, kitting, language translation and worldwide distribution. Leave it to us, and you'll have more time to do the things you like best. And that's no fish story. Call us for a free catalog.
1-800-448-6737
Putting Imagination To Work
digital
© Digital Equipment Corporation, 1994. The DIGITAL Logo is a trademark of Digital Equipment Corporation.


POWERBUILDER
DEVELOPERS WANTE
We're looking for senior application developers and technical arc
join our rapidly expanding client/server team. To succeed, you will
relevant experience which must include at least 6 month
PowerBuilder development within an SQL compliant relat
Wellington Associa
Fax: 800-555-83
Looking
CLIENT/SERVER
EXPERIENCE
Unique opportunities are available
for programmers experienced in
implementing Client/Server systems.
Applicants must have a four-year
technical degree, four to six years'
development expe e, and profi-
ciency with bo racle and
PowerBuilder. Be
tuition
PROGRAMMERS/ANALYSTS
Information consulting firm has opportunities throughout the East as well as in California
for professionals with strong technical backgrounds in system design, development
and methodology, as well as experience with Windows, DB2 and PowerBuilder.
Benefits include exc K, continuing ed., and much more.
ment Services
98
ANALYSTS/
PROGRAMMERS
PowerBuilder P/A's OPEN
ORACLE/SYBASE DBA's 50-60k
SMALLTALK 40-60k
HOGAN P/A's 45-65k
IMS/DB2/IDMS 40-50k
EDP Auditors 40-50k
UNIX/C/C++/OOP 40-50k
IDEAL/DATACOM 40-50k
ORACLE V6.0 & 7.0 40-50k
AS/400 S/P's P/A's 35-45k
MVS/CICS/COBOL 35-45k
For additional information on
these and nationwide opportunities,
fax resumes in confidence to:
CAREERS
UNLIMITED
800-555-3999
POWERBUILDER
PROGRAMMERS
Looking for seasoned
PowerBuilder programmers
for a major development
effort. Environment includes
Novell LAN, SQL Server
running on NT. Interested
primarily in independent
developers but will consider
companies. Need is urgent.
Please fax interests and back-
ground to (800) 555-2222.
Computer Network
800-555-2222
ARE APPLICATION MANA
ompany needs professional to manage a team of dBASE
orking on client/server applications for end users in our
dministration department. Candidates should have 4 plus
anagement experience, expertise in database application
development, and proficiency in PowerBuilder.
HARTMAN & COMPANY
800-555-6318

*(continued from page 46)*

ible analysis tool. The flexibility of WPT versus the FWT can be compared to that of having a complete set of sockets for a ratchet rather than a single socket to attach to it. The ratchet (algorithm) works the same regardless of the socket (basis) that is chosen. The flexibility of the tool is in choosing the appropriate socket (basis) for the particular nut (problem). The choice of wavelet and scaling functions is then analogous to selecting from English, metric, or Torx socket sets for use with the ratchet. The WPT allows tailoring of the wavelet analysis to selectively localize spectral bands in the input data as well as to correlate the signal to the wavelet. Not only can the best wavelet be chosen to analyze a particular signal but the best orthonormal basis can as well. In signal-processing terminology, the various bases of the wavelet packet transform can be used as arbitrary adaptive tree-structured filter banks.

### Piece-wise Convolutions and Traversing the Tree

The implementation of the WPT is itself a generalization of the FWT routine presented in my previous article. As with the FWT, the kernel operations are the decimating and interpolating convolutions, as presented in Listing One (page 101). The convolutions performed are actually piece-wise convolutions, due to the discrete nature of the data. The routine *DualConvDec2* replaces the routines *ConvolveDec2* and *Dotp* in the FWT code. The routine *DualConvInt2Sum* replaces *ConvolveInt2*, *DotpOdd*, and *DotpEven* in the inverse FWT code.

Both convolution routines are designed to operate upon aperiodic data of finite length. The data does not represent an infinitely repeating pattern and is assumed to be surrounded by zero-valued data; see Figure 5. To support this data model, each data array

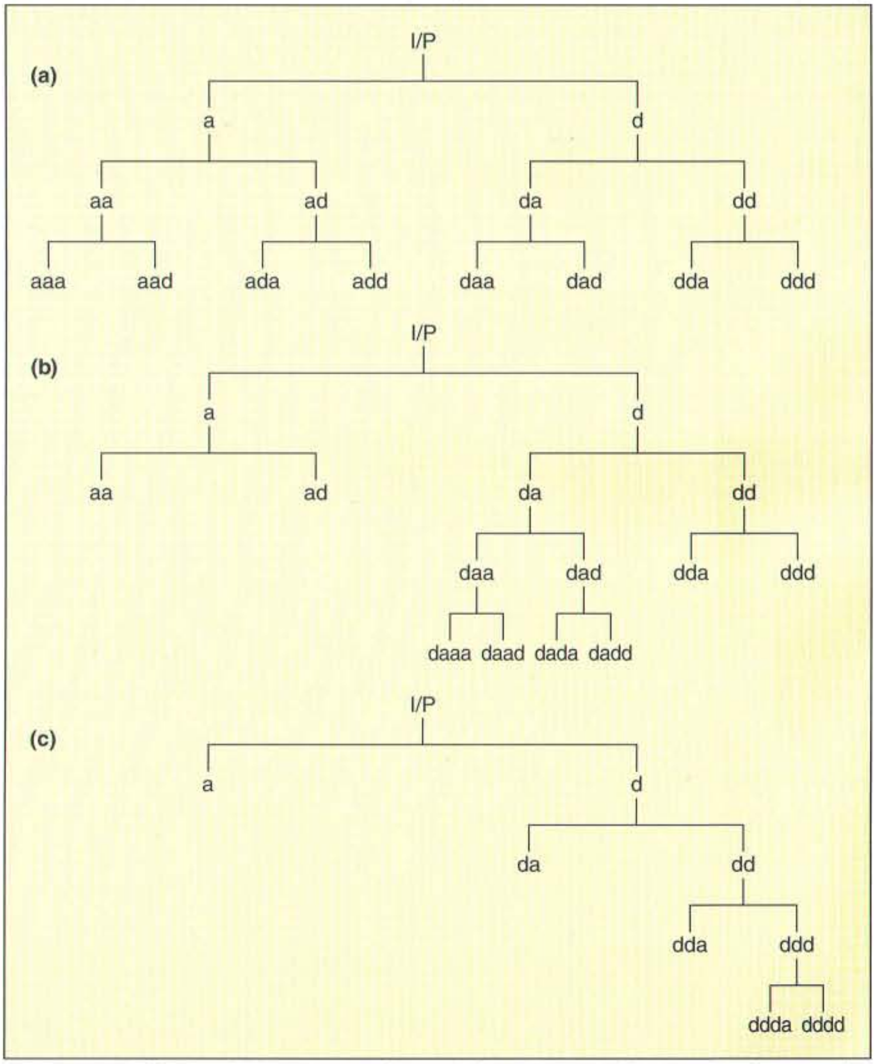



***Figure 4:*** *Different disjoint covers formed from the WPT binary tree (Figure 3) yield different wavelet packet bases. (a) A subband basis; (b) an orthonormal basis subset; (c) a basis which is the opposite of the wavelet basis (better frequency localization at higher frequencies).*

Introducing IBM C Set ++ V2.1 and KASE:Set.

C Set ++ V2.1 including Kaseworks KASE:Set. Starting at $393(U.S.) - CD-ROM Version.

# Object oriented development.

C Set ++ Version 2.1 for OS/2® from IBM Software Solutions is one of the most complete object-oriented application development packages you can buy.

# Plus.

Its 32 bit C/C++ compiler, with its advanced code optimizer *and* WorkFrame/2 V2.1 - a completely new development environment based on OS/2's object-oriented Workplace Shell - allows you to create up-to-the-minute mission critical applications.

You also receive a specially tailored copy of Kaseworks KASE:Set, a visual design and code generation tool which allows you to build visual applications even more quickly, with a minimal investment in learning time.

# Plus!

C Set ++

**C Set ++ Technical Features**

- Standards conforming C/C++ compiler.
- Optimizer for i386, i486 and Pentium™ processors.
- Extensive C++ Class Libraries plus NEW drag/drop support.
- Visual PM Debugger with object-oriented features.
- Graphical C++ Class Browser.
- Visual Trace Analysis tool.
- Precompiled headers.
- Developer's ToolKit new V2.1 with tools for multimedia and SOM applications.

**To order C Set ++ for OS/2 including KASE:Set, or for further information call 1-800-342-6672 (U.S.A.) or 1-800-465-7999, ext. 676 (Canada). Or contact your local IBM software dealer.**



IBM®

**CIRCLE NO. 737 ON READER SERVICE CARD**

*(continued from page 50)*
is appended with additional data storage equal to the length of the wavelet filter, minus one (the shaded elements). The extra data is filled with the terminating convolution values as the wavelet filters "slide off" the end of the data set. Extending the convolution data by this additional amount during the decomposition process of the forward transform ensures perfect reconstruction of the original input data by the inverse transform. *DualConvDec2* also uses partial dot products at both ends of the convolution to simulate the implied zero-valued data (the dotted lines) outside of the data array. The dotted lines on the filter elements indicate coefficients not used in the partial dot product calculations. *DualConvInt2Sum* does not need to do this, since all information necessary for reconstruction is contained within the extended data arrays. Note that *DualConvInt2Sum* performs only the odd-valued dot product at the beginning of the convolution for the initial, reconstruction data point.

*The discrete wavelet transform is actually a subset of a far more versatile transform, the wavelet packet transform*

The WPT data is stored in the structure *WPTstruct*, defined in Figure 6(a). The structure contains storage for the number of levels in the transform, the length of the original, untransformed data array, and a pointer to a two-dimensional matrix of data arrays. The size of the matrix in dependent upon three factors. These are the number of levels in the transform, the length of the original data array, and the length of the transform filters. The length of the data array is itself affected by the length of the transform filters; see Figure 6(b).

Figure 6(c) shows the matrix structure as the wavelet packet binary tree. The data-array pointers are allocated memory from the heap as necessary to form the disjoint cover for the chosen orthonormal basis. Those array pointers not required for the disjoint cover are set to zero. The example shown in the figure represents a three-level wavelet basis for an input of 40 data points with transform filters containing six coefficients.

The routine *DualConvDec2* is used by *AWPT*, the forward wavelet packet transform routine; see Listing Two (page 101). *AWPT* accepts pointers to the input data array, the WPT data structure, and the transform filter arrays, and the length of the filters. The transform routine works down the levels of the binary tree performing convolutions on the data arrays. Each level of the binary tree is traversed by taking adjacent pairs of array pointers as the destination nodes for the low-pass and high-pass convolutions. If the array pointer for the low-pass convolution is zero, the convolutions are not performed since the destinations are not part of the current disjoint cover.

At the highest level (where *i* equals 0) the input data array is the source for the convolutions. On each subsequent level, the sources are the arrays on the previous level. The appropriate array in the binary tree is determined by dividing the current *j* index by 2. After each convolution operation, the data length is divided by two, in order to keep track of the effect of the decimation operation during the convolutions.

*DualConvInt2Sum* is used by *IAWPT*, the inverse wavelet packet transform routine. *IAWPT* accepts pointers to the source WPT data structure, the output data array, the transform filter arrays, and the length of the filters. The inverse transform routine works up the levels of the binary tree, performing convolutions on the data arrays and reconstructing the higher-resolution data on each level. Each level of the binary tree is traversed in the same fashion as was *AWPT*. The destination arrays are on the next-higher level of the matrix, and they are selected by dividing the *j* index by 2. At the highest

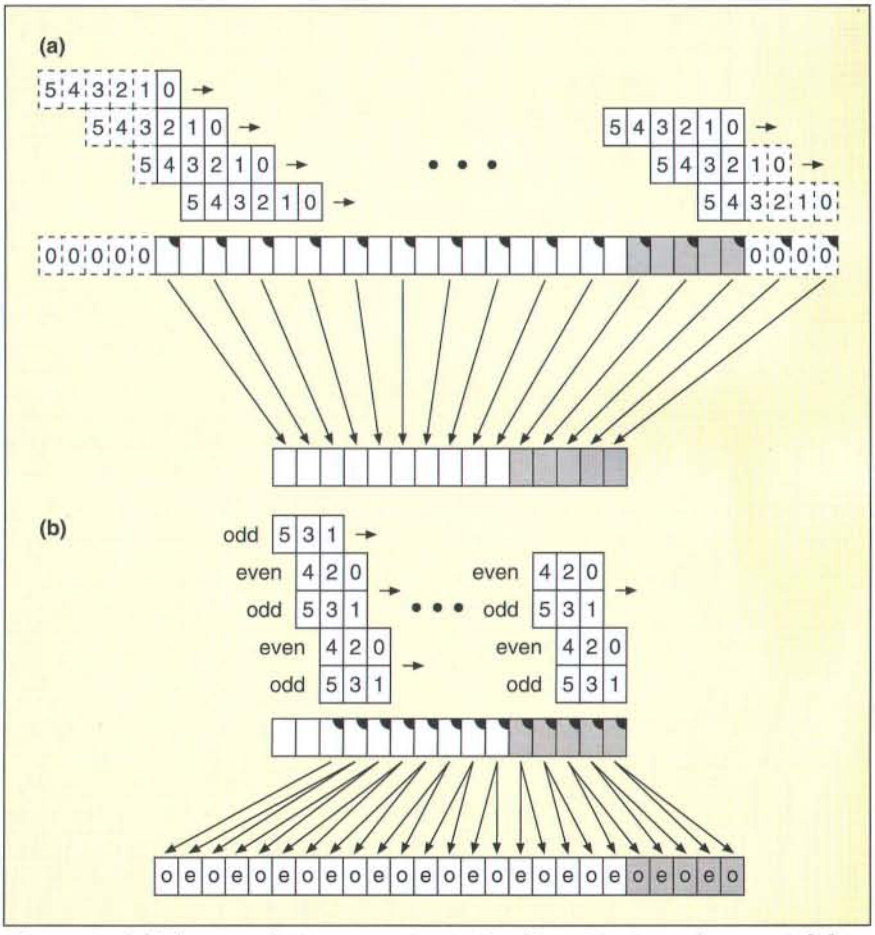
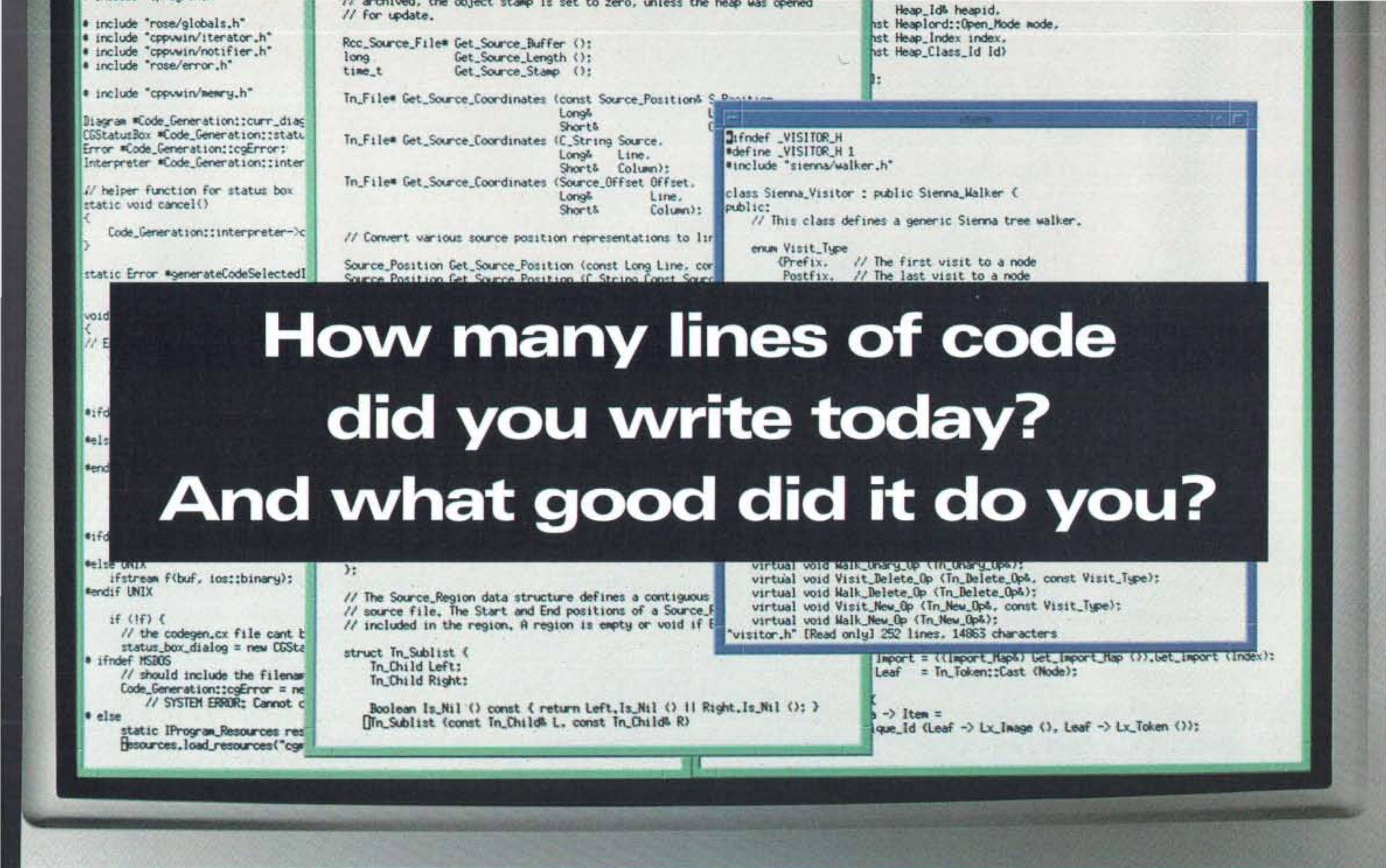



***Figure 5:*** *(a) The convolution operation in* DualConvDec2 *employs partial dot products at both ends of the data array to simulate zero-valued data surrounding it; (b) the convolution operating in* DualConvInt2Sum *performs dot products with alternating odd and even filter components to simulate interpolation. The additional convolution data generated by* DualConvDec2 *is used to accomplish perfect reconstruction. Marked data elements indicate starting points of dot-product calculations.*



# How many lines of code did you write today? And what good did it do you?

## Presenting The Rational Approach, a new line of thinking in software development.

This is not one of those "code is a four-letter word" ads. Anything but, actually. We know how important – and all too often, how unappreciated – developers and the product of their labors can be.

It's just that we think their efforts could be utilized so much more efficiently. After all, who needs software that's late, over budget, buggy, and with far less functionality than required?

Our solution is built around what we call The Rational Approach™. And it's a lot more than just another collection of tools and consultants. From technology assessment through the deployment of a full object-oriented system, we have the people, the products, the services, and perhaps most importantly, the experience to deliver a truly complete solution.


*We make it our business to ensure the success of yours, with complete software engineering solutions.*

*Watch the productivity of your development team improve with The Rational Approach.*

A key component of our solution is Rational Rose/C++, a graphical tool that automates use of the Booch '93 method and generates your C++ source code. Rational Rose/C++ is available for Windows™ PCs and UNIX® workstations.

The Rational Approach can help you achieve the real-world benefits of object-oriented technology and iterative software development: increased productivity, software and design re-use, and shorter time to market.

For more information and a free copy of our new white paper on The Rational Approach, call us today at **1-800-767-3237.** Quite simply, it's the logical thing to do.

RATIONAL
Software Engineering Solutions

**CIRCLE NO. 785 ON READER SERVICE CARD**

Rational, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951 (408) 496-3600. France +33-1-47-17-41-77; Germany +49-89-797-021; U.K. +44-273-624814; Sweden +46-8-761-0600; Australia +61-3-521-0507; Taiwan +886-2-720-1938. Representatives: Italy +39-2-264-0107; Spain +34-1-579-21-13; Israel +972-3-575-5550; Japan +81-03-3779-2541

*(continued from page 52)*
level (*i* equals 0), the destination array is the output data array. After each convolution operation, the data length is doubled to keep track of the effect of the interpolation operation during the convolutions.

The code listings presented here are written in ANSI C and have been tested with Borland Turbo C 2.0. They should compile on any compiler that is compliant with ANSI C. I've also written a wavelet packet transform demonstration program which is available electronically; see "Availability" on page 3. The electronic version includes the demo program, sample data files, support drivers, and documentation.

## Conclusion

The wavelet packet transform generalizes the discrete wavelet transform and provides a more flexible tool for the time-scale analysis of data. All of the advantages of the fast wavelet transform are retained since the wavelet basis is in the repertoire of bases available with the wavelet packet transform. Given this, the wavelet packet transform may eventually become a standard tool in signal processing.

## References

Cody, Mac A. "The Fast Wavelet Transform." *Dr. Dobb's Journal* (April 1992).

Coifman, Ronald R., Yves Meyer, and Victor Wickerhauser. *Wavelet Analysis and Signal Processing.* New Haven, CT: Yale University, 1991, preprint.

**DDJ**

**(Listings begin on page 100.)**

To vote for your favorite article, circle inquiry no. 5.

(a)

```
typedef struct
{
  short levels, length;
  REAL_TYPE ***DATA;
} WPTstruct;
```

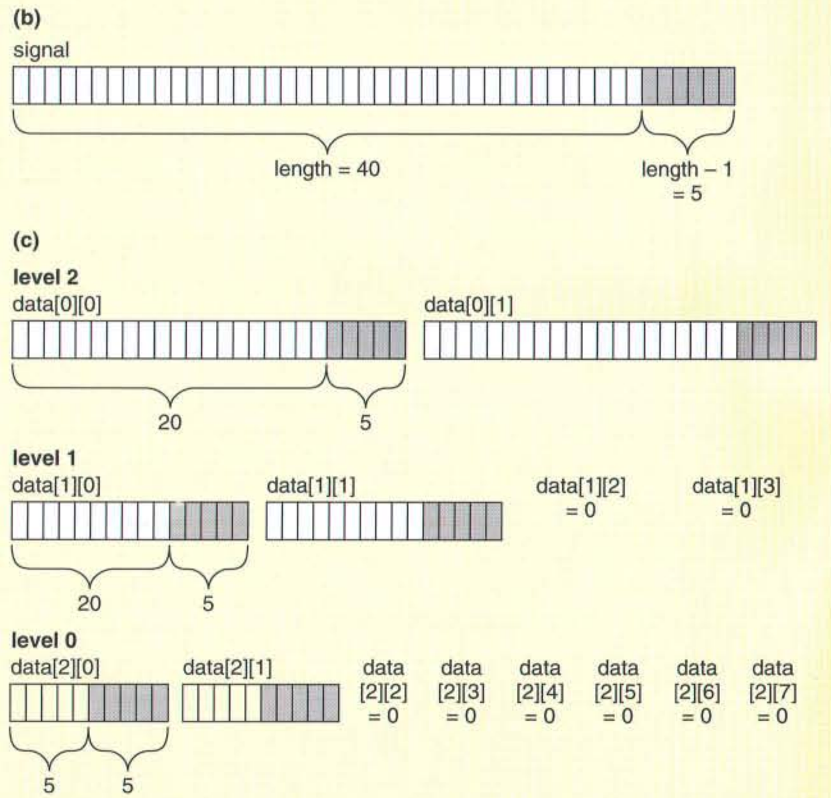
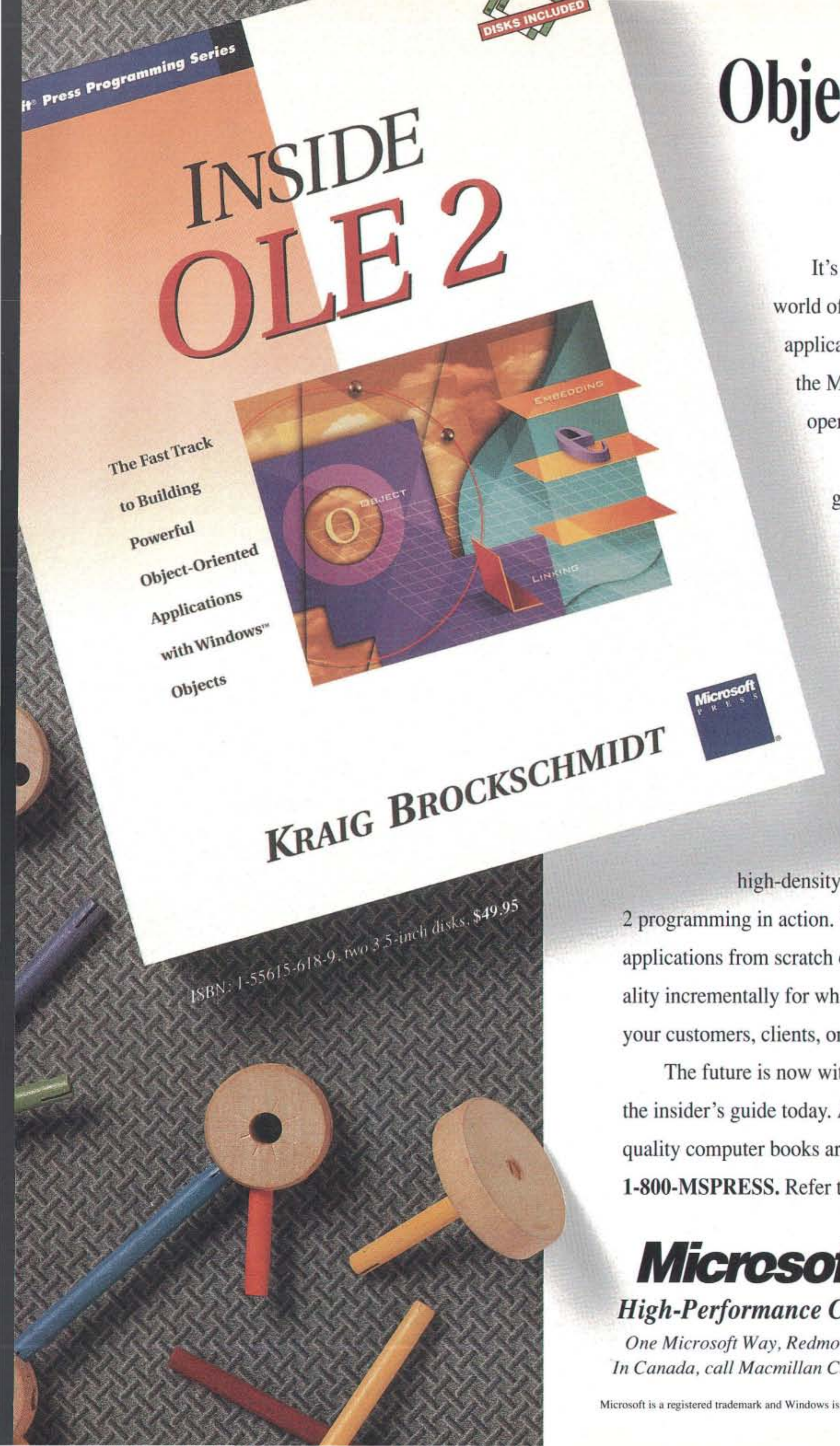



***Figure 6:*** *(a) The data structure for the WPT. The* REAL_TYPE *declaration can be defined either as* float *or* double; *(b) the storage structure for the original signal consists of the data, plus appended storage equal to the filter length, minus 1; (c) the data component of* WPTstruct *is a two-dimensional matrix of data arrays forming the wavelet packet binary tree. Data-array pointers set to 0 indicate parts of the tree that aren't part of the disjoint cover (the wavelet basis).*




# Objectivity

It's a giant leap into the world of object-oriented application development for the Microsoft® Windows™ operating system.

And we do mean giant. You get more than 1,000 pages of expert advice from one of the best programming minds. Plus, over 50,000 lines of practical source code examples in C/C++ on two high-density disks let you see OLE 2 programming in action. You'll learn to build applications from scratch or add OLE 2 functionality incrementally for whatever makes sense for your customers, clients, or business.

The future is now with OLE 2, so get the insider's guide today. Available everywhere quality computer books are sold, or call **1-800-MSPRESS.** Refer to ad ADJ.

**Microsoft Press**

***High-Performance Computer Books***

*One Microsoft Way, Redmond, WA 98052-6399*
*In Canada, call Macmillan Canada, 416-293-8141*

# Fuzzy Logic in C: An Update

*Completing a fuzzy-based inference engine*

John A.R. Tucker, Phillip E. Fraley, and Lawrence P. Swanson

Early last year, we were looking for a software implementation of fuzzy logic. Greg Viot's article "Fuzzy Logic in C" (*DDJ*, February 1993) was a step towards what we needed, but it didn't include the necessary initialization, parsing, and output functions. Consequently, we filled in the gaps by writing functions that, together with Greg's code, make a working fuzzy-logic program you can use. Listing One (page 101) is the complete source code for the updated version (which includes Greg's original code and our additions). The enhancements, which we'll focus on in this article, are shaded as well as identified in the comments. For background on fuzzy logic in general and Greg's techniques in particular, refer to his original article.

### Rule Files and Structures

We saw right away that the parsing of the rules file would be a problem to generalize for all possible combinations of antecedents and consequences, so we elected to simplify the problem by allowing only two antecedents and one consequence.

The generalized case would have resulted in loss of clarity. We didn't try to be clever about our functions; in fact, they are quite direct (three segments are repetitive). The extensive use of linked lists and pointers to structures in *initialize_system()* related to the rules are, however, quite involved. Nor did we optimize or generalize the code. This makes it possible for you to modify the code to accept other input files by copying existing code segments and making minor adjustments. Finally, we took full advantage of understanding the input data structures for the specific example of the inverted-pendulum problem Greg described.

To allow for easy alteration of the fuzzy sets or rule definitions, we used three ASCII files with fixed names and formats as the input files that describe the fuzzy sets (angle, velocity, and force). Similarly, an ASCII file is used to describe the rules file. These four files are to be located in a common directory from which the program is run.

In the three files describing the fuzzy sets (in1, in2, and out1), you can use any name ten characters or less in length on the first line as a name for the input fuzzy set. The first column of the subsequent lines is for the name of the membership element of that fuzzy set, again limited to ten characters. The next four columns describe the corner points of the membership (if the third and fourth columns are the same, the shape is a triangle). White-space, spaces, or tabs separate the columns. You may have as many rows of membership elements as you please, but five, seven, or nine seem to be the best choices. Take care not to include any blank rows.

The first file, in1 (angle), looks like Figure 1. The files in2 (velocity) and out1 (force) are similar. In *initialize_system()*, we have three nearly identical code fragments. You can block copy them and make the few changes required. The cycle is as follows: Open the file, set a pointer and allocate memory, read the fuzzy set's name, read a line of data from the file, set a pointer to the next structure, assign values to the structure elements, and lastly, close the file. The differences in these three segments are in lines 1 and 2 (the filenames are different), lines 5 and 6 (the pointers point to differing places), and lines 27 and 33, where the filenames in the error messages are different.

We included an error trap to detect if either *slope1* or *slope2* is less than or equal to 0, a condition not allowed in the original program. If such an error is encountered, the program exits with appropriate information. The setting of the pointers for the rules file is more complex. In the original article, Greg suggested a file that looked like Figure 2(a). Although we liked the form of this file, it was complex to parse so we stripped the file to its essential elements: the name of the fuzzy set elements and the order in which they appear in each rule; see Figure 2(b). We used an awk and sed pipeline to strip the "rule" file and create a more suitable form for parsing in a file named "rules." (The command line *awk '{print $6, $10, $14}' rule|sed 's/)//g'> rules* does this elegantly. You can create the rules file di-

```
Angle

NL    0    31   31   63
NM   31    63   63   95
NS   63    95   95  127
ZE   95   127  127  159
PS  127   159  159  191
PM  159   191  191  223
PL  191   223  223  255
```

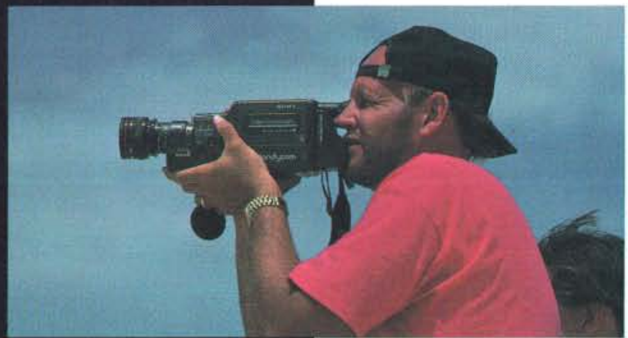
***Figure 1:*** *The in1 file; values can be altered as desired.*

*John teaches computer courses at Albright College and Reading Area Community College; Phillip is working on several projects, including proton models, large color images, and neural networks; Lawrence currently works as a test engineer. They can be reached through the* DDJ *offices.*

# NEW VERSION 3.0
# NEXPERT *OBJECT*®
**Building Smarter Applications™**

*"NEXPERT has effectively captured the design engineer's knowledge required to perform CCD simulations, significantly improving productivity at SONY."*

SONY



**SHORTENED TIME TO MARKET FOR CAMCORDERS**
NEXPERT is the core of SONY's simulation system for speeding up the design and testing of highly-specialized chips used in video cameras. The system, called XAS, reduced design time by two thirds. XAS runs in a local area network of Sony NEWS workstations, using X Windows, with more than 500 complex simulation rules distributed across 15 NEXPERT knowledge bases.

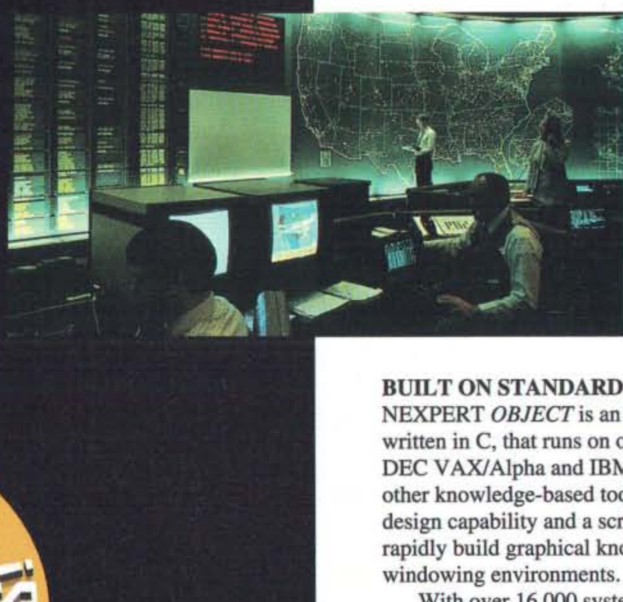
*"By adding knowledge to existing applications, NEXPERT has allowed us to leap beyond our already strict audit control standards to help us perform a difficult and critical task."*

Chemical Bank


**RISK MANAGEMENT**
Chemical Bank uses NEXPERT for daily review of over a billion dollars in worldwide foreign exchange transactions. The Digital VAX-based application called Inspector, ties to Oracle databases, C programs, and a communications network spanning 23 countries. Given the dollar amounts involved in transactions, once Inspector identified even one fraudulent trade, it paid for itself many times over.

*"We've recognized that knowledge is a resource that can be harnessed and leveraged to strengthen productivity and reduce costs in many areas of our business — and NEXPERT is the key."*

BCTel



**CUSTOMER SERVICE**
BCTel, Canada's second largest telephone company, uses NEXPERT to streamline business practices and help generate, recover, and protect revenue. NEXPERT is at the heart of several multiplatform applications ranging from network overload management to customer services and billing monitoring. The systems run on Sun and Digital UNIX workstations, Digital VAX/VMS systems and PCs.


**BUILT ON STANDARDS**
NEXPERT *OBJECT* is an object-and rule-oriented development tool written in C, that runs on over 35 platforms, from PCs and Macs to Unix, DEC VAX/Alpha and IBM mainframes. NEXPERT extends beyond other knowledge-based tools with functionalities such as a robust GUI design capability and a script language. Developers can now more rapidly build graphical knowledge-based applications that support all windowing environments.

With over 16,000 systems in use worldwide and an extensive network of solution providers and systems integrators, NEXPERT has emerged as the proven standard for developing knowledge-based systems.

**Call us today at 415-321-4488 or 1-800-876-4900 ext. 804 to register for our free seminars or to receive additional documentation.**

CIRCLE NO. 602 ON READER SERVICE CARD

*Object-Oriented Tools for Portable Client/Server Development™*

NEURON DATA

*(continued from page 56)*
rectly, as with the input files, and eliminate the clearer representation of the rule file entirely if you do not have these tools.) You can have more or less than the 15 rules in Greg's article. Add or delete them as you please, one rule per row. Be certain that membership-element names are exact matches in all the files, including the rules file. In particular, note that upper and lower case are *not* equivalent.

Once *initialize_system()* is written, you're limited to two inputs and one output. You will need to make changes to the arguments for *fscanf()* and define new buffers to accommodate any other combination.

The insight to the rules structures initialization is that structures of *rule_type* and *rule_element_type* form the acceptable rules at the time of initialization. That is, appropriate fuzzy inputs (antecedents) are associated (linked) with a fuzzy output (consequence) as defined in the rules at the time the rules file is read. Values in the *mf_type* structure are pointed to by the pointer stored in the *rule_element_type *value*. Later, if a 0 is pointed to by any of the *if_side* value pointers, the function *defuzzification()* will equate to 0, and subsequent calculation of the *sum_of_products* and *sum_of_areas* will not be affected. See Figure 3 for the complete relationship of all the data structures and their pointers.

```
(a)  rule 1: IF (angle is NL) AND (velocity is ZE) THEN (force is PL)
     rule 2: IF (angle is ZE) AND (velocity is NL) THEN  (force is PL)
     rule 3: IF (angle is NM) AND (velocity is ZE) THEN (force is PM)
     :
     rule 15: IF (angle is PL) AND (velocity is ZE) THEN (force is NL)

(b)  NL ZE PL
     ZE NL PL
     NM ZE PM
     :
     PL ZE NL
```

***Figure 2:*** *(a) Original rules file; (b) modified rules file.*

### Using the Updated Program

To illustrate how you can use the updated fuzzy-logic program, we'll refer you to the rule in Figure 4, where we begin by opening the rules file, allocating memory for a structure *rule_type* and setting a pointer to it, and scanning the rules one line at a time. As each line (rule) is read, we "know" that the first field in the line is the angle (structure *io_type*, pointed to by **membership_functions*), so we begin searching its fuzzy-set members (structures *mf_type*), doing a string match on the membership element name, NL. When the match is found, memory for a *rule_element_type* structure is allocated, the *address* to the *value* element of the matching *mf_structure* is stored, and a pointer to *rule_type*, pointed to by the **if_side*. A pointer to the second field (in this case, velocity) is also established as a pointer (element **next*) in *rule_element_type*.

The second field of the rule (velocity) is then used to search for a string match on its membership-element name, ZE, in the second *io_structure *membership_function*, and a pointer to the *address* where its value is located is stored in the next *rule_element_type *value*. Finally, the last element of the rule, the

*(continued from page 58)*

consequence (force, in our example), is treated in the same manner: The *address* of the match is stored in the *rule_element_type *value* pointed to by the *rule_type *then* side when the appropriate membership element name, PL, is matched.

These steps are repeated for every rule in the rules file; refer again to the first three rules in Figure 3.

To complete the alterations, other changes included placing the two anchor pointers *System_Output* and *System_Inputs* as global pointers along with the existing *Rule_Base*, adding macro definitions for max and min for cross-compiling onto MS-DOS platforms, and adding the *#include* for the function *strcmp()*; see Example 1. We also included the necessary function to accept two inputs from the command line as arguments for the initial condition *get_system_inputs()* and a function *put_system_outputs()* to examine the exit status of a single inference pass on the input data.

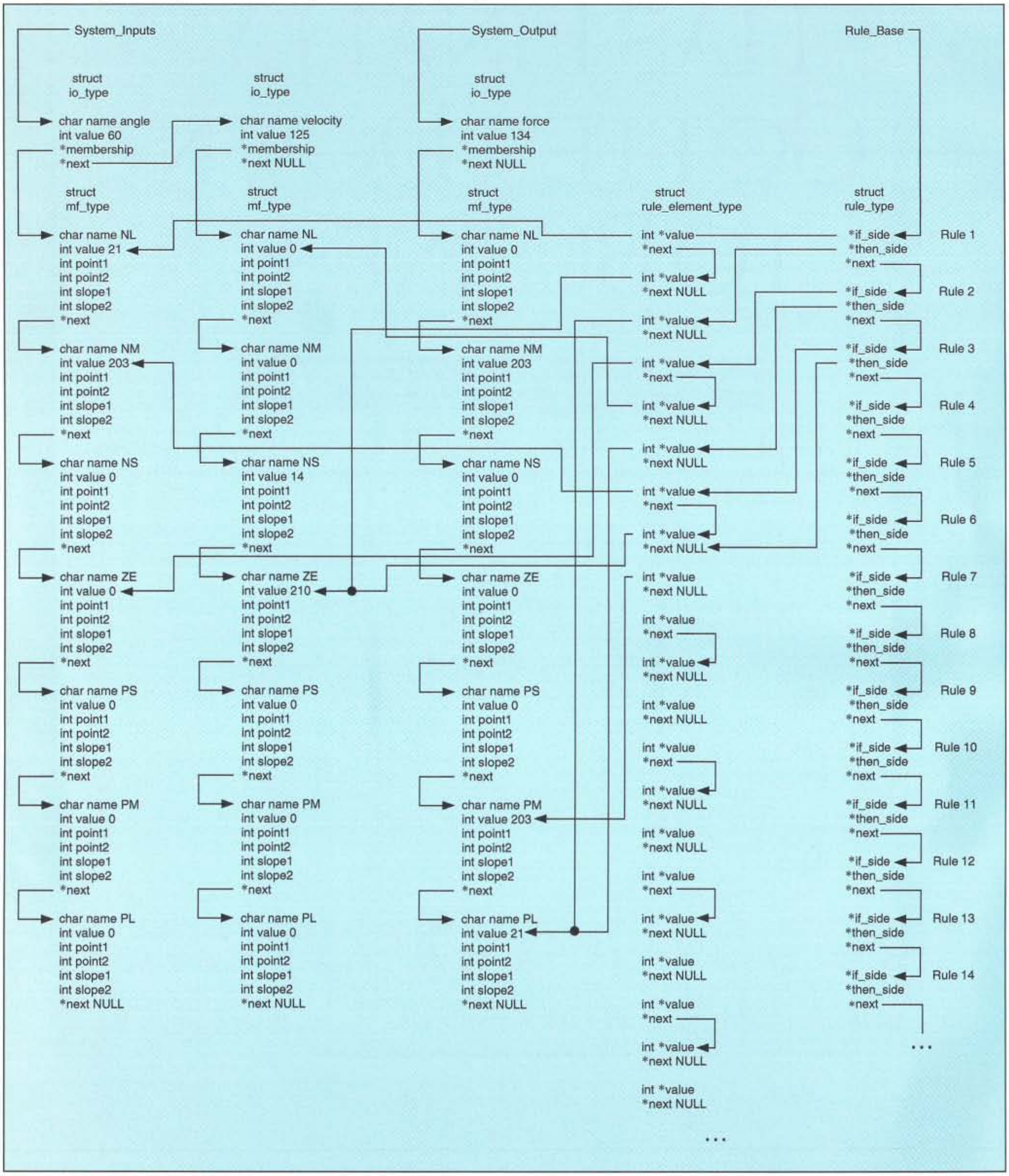



***Figure 3:** Relationship of data structures and their pointers.*

```
rule 1: IF (angle is NL) AND (velocity is ZE)
                               THEN (force is PL)
```

***Figure 4:** Sample rule used to develop Figure 3.*

```
fuzz 60 125
angle: Value=60
  NL: Value 21 Left 0 Right 63
  NM: Value 203 Left 31 Right 95
  NS: Value 0 Left 63 Right 127
  ZE: Value 0 Left 95 Right 159
  PS: Value 0 Left 127 Right 191
  PM: Value 0 Left 159 Right 223
  PL: Value 0 Left 191 Right 255
velocity: Value=125
  NL: Value 0 Left 0 Right 64
  NM: Value 0 Left 31 Right 95
  NS: Value 14 Left 63 Right 127
  ZE: Value 210 Left 95 Right 159
  PS: Value 0 Left 127 Right 191
  PM: Value 0 Left 159 Right 223
  PL: Value 0 Left 191 Right 255
force: Value=134
  NL: Value 0 Left 0 Right 63
  NM: Value 203 Left 31 Right 95
  NS: Value 0 Left 63 Right 127
  ZE: Value 0 Left 95 Right 159
  PS: Value 0 Left 127 Right 191
  PM: Value 203 Left 159  Right 223
  PL: Value 21 Left 191 Right 255
  Rule #1: 21 210 21
  Rule #2: 0 0 21
  Rule #3: 203 210 203
  Rule #4: 0 0 203
  Rule #5: 0 210 0
  Rule #6: 0 14 0
  Rule #7: 0 0 0
  Rule #8: 0 210 0
  Rule #9: 0 0 0
  Rule #10: 0 210 0
  Rule #11: 0 14 0
  Rule #12: 0 0 203
  Rule #13: 203 210 203
  Rule #14: 0 0 0
  Rule #15: 0 210 0
```

***Figure 5:** Output generated with a scaled angle of 60 and scaled velocity of 125.*

```
#include      <string.h>
#define max(a,b) (a<b ? b : a)
#define min(a,b) (a>b ? b : a)
struct io_type *System_Inputs;
struct io_type *System_Output;
```

***Example 1:** Adding the* #include, *global pointers, and macros.*

*(continued from page 60)*

After using the code with various inputs, we needed to add error traps because we were getting core dumps with certain input. These were caused by division by zero when there were no rules in the set to cover the condition. Consequently, we added the code in Example 2(a) to the original function *defuzzification()*. We also added Example 2(b) to *rule_evaluation()*.

*After using the code with various inputs, we needed to add error traps because we were getting core dumps with certain input*

To further illustrate how you can use the program, assume the scaled angle of 60 and a scaled velocity of 125 as in Figure 5. The line *force: Value=134* reflects the defuzzified and scaled single-valued output for the two inputs. It would be instructive to interface this program to a graphics output device where a loop could be created and the inverted pendulum balanced. Alternatively, a batch file or shell script could feed new inputs and use the output to generate the two new inputs, storing intermediate data in a file. Or, you might try graphing the trapezoidal output areas made on each iteration.

**DDJ**

**(Listing begins on page 101.)**

To vote for your favorite article, circle inquiry no. 6.

```
(a)   if(sum_of_areas==0)
      {  printf("Sum     of Areas = 0, will cause div error\n");
         printf("Sum     of Products= %d\n",sum_of_products);
         so->value=0;
         return;
      }

(b)   int nomatch=0;
      for(tp=rule->then_side;tp!=NULL;tp=tp->next)
      {  *(tp->value)=max(strength,*(tp->value));
         if(strength>0)nomatch=1;
      }
      }
      if(nomatch==0)printf("NO MATCHING RULES FOUND!\n");
```

***Example 2:** (a) Code added to the original function* defuzzification( )*; (b) code added to* rule_evaluation( ).

# *Introducing EasyCASE® 4.1 for Windows™ Workgroup Edition.*

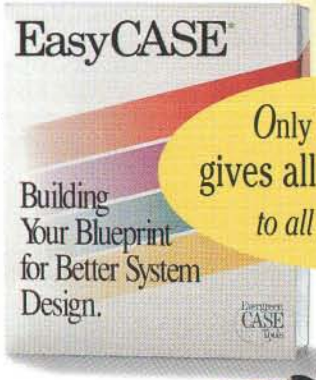
*D*esigning complex systems with teams can spawn its own set of challenges. But with the new EasyCASE Workgroup Edition, you can afford to give your *entire department* the tools it needs to build easily maintainable systems. Based on EasyCASE 4.0 for Windows–*Data Based Advisor* magazine's Best CASE/Design Tool for 1993–EasyCASE Workgroup Edition offers you concurrent, multi-user access to shared projects on your network server.

★1993★ DATA BASED ADVISOR Readers Choice Award

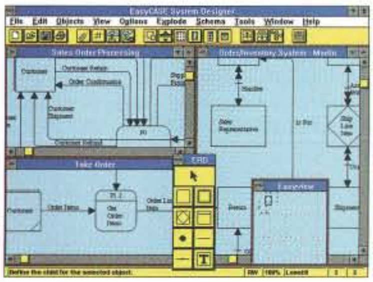
***Breed sound design and security, team-wide.***
You'll get locking at the chart, data-dictionary record level and project-wide level; user passwords, permissions; read-only and lockout modes; plus access control.

***Develop client/server database systems with the schema generator.***
You won't pay extra for our schema generator. Use it for xBASE(dBASE III/IV, FoxPro, Clipper, Paradox), plus a wide range of SQL databases (Oracle, Ingres, Informix, XDB, Progress, DB2, Sybase, SQL Server, Watcom SQL, SQLBase, etc.) You can even reverse engineer xBASE. EasyCASE is also helpful in client/server application development.

***Arm your team with the most methodologies.***
EasyCASE gives you the *largest* selection of structured analysis, design and data modeling methodologies available in *one* tool: Yourdon/DeMarco, Gane & Sarson, SSADM, Ward-Mellor, Hatley, Yourdon/Constantine, Martin Information Engineering, Shlaer-Mellor, Chen, IDEF1X, and Merise. Plus a powerful data dictionary repository with extensive reporting capabilities.




***Easy to master in Microsoft® Windows.™***
Enjoy a consistent graphical user interface; object-oriented editing; easy chart linking; rules; integrated data dictionary; reports and analysis. First-time users become productive quickly with a tutorial, on-line help, methodology guide and sample projects. Also available: EasyCASE 4.0 for MS-DOS® (single user version only).



***The highest quality systems for the lowest cost per seat.***
EasyCASE is the undisputed price/performance leader. No other product in its category gives you so much value, power, flexibility and return on investment as EasyCASE Workgroup Edition. Over 12,000 users worldwide rely on EasyCASE to assure higher quality real-time and information systems. Fact is, you can't afford not to have it.

MICROSOFT® WINDOWS™ COMPATIBLE

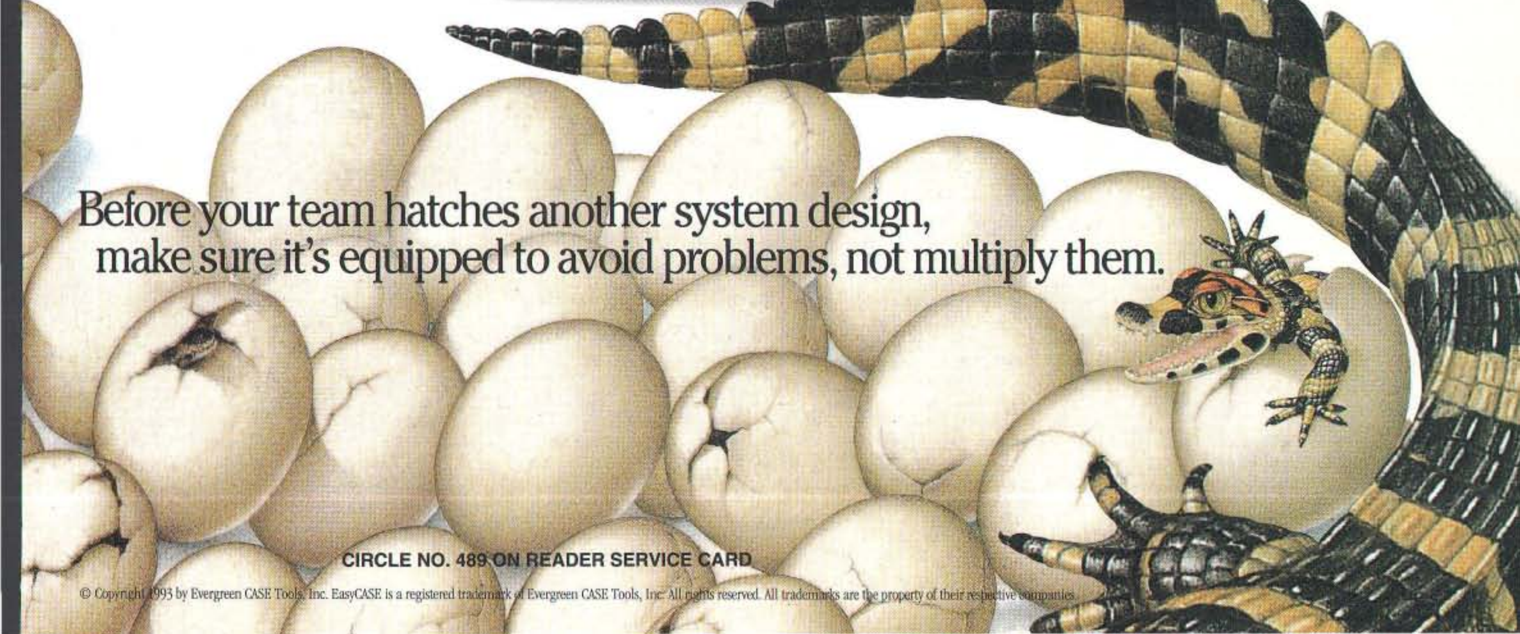
***Look for these features coming soon.***
Enhanced client/server support, reverse engineering of SQL, links to PowerBuilder™ and SQLWindows,™ object-oriented methods, version control, 32-bit version and OLE. Call for availability.

Evergreen CASE Tools

*"EasyCASE's exceptional charting tools, and numerous supported process and data methodologies put the product well above the pack for products in this price category...The combination of useful CASE tool functionality and low price makes EasyCASE 4.0 a winner!"*—*PC Week, July '93*

Only EasyCASE gives all this **power** *to all your people.*

*For price and product information, call:*
*1-800-929-5194*
*Outside the U.S., call (206) 881-5149*
*8522 154th Ave. NE, Redmond, WA 98052*
*FAX:(206) 883-7676*



Before your team hatches another system design, make sure it's equipped to avoid problems, not multiply them.

CIRCLE NO. 489 ON READER SERVICE CARD

# Digital I/O with the PC

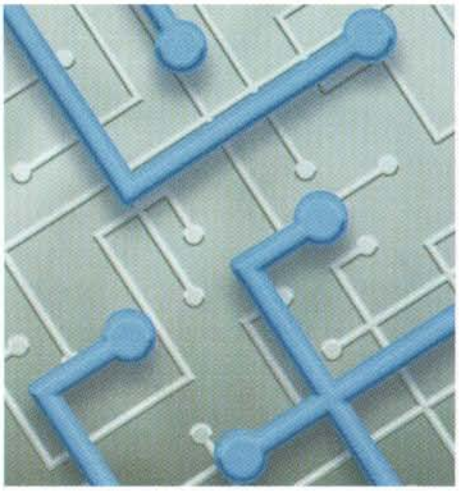
## *Putting the parallel port to work*

Brian Hook and Dennis Shuman

Data-acquisition and analysis is often performed with dedicated, proprietary, and expensive laboratory instruments. However, the PC's open architecture makes it a cost-effective alternative for many data-acquisition and analysis projects. One particular project we developed at the Acoustic/Electronic Insect Detection Laboratory at the United States Department of Agriculture, Agricultural Research Service facility in Gainesville, Florida required just such a system. The system is an integrated hardware/software setup that allows for digital input and output in a low-end PC configuration. This article describes what we learned while implementing digital I/O via the PC's parallel port.

The system, known as "EGPIC" (Electronic Grain Probe Insect Counter), checks for insects in stored-grain bins and elevators. It does this by electronically sensing insects that crawl into specially designed probes placed at a number of locations in the grain mass. The hardware side of the EGPIC is responsible for detecting an insect and generating the appropriate signal for some digital-input computer interface. The software is responsible for reading, analyzing, displaying, and storing the collected data. We selected the PC as the host system for the software because of its wide, low-cost availability and profusion of development tools.

*Brian is a programmer at the USDA developing data-acquisition and analysis software. He can be reached on the Internet at bwh@cis.ufl.edu or on CompuServe at 72144,3662. Dennis, a research scientist at the USDA, develops electronic/accoustic systems to detect insect pests in agricultural commodities. He can be reached at USDA, ARS, 1700 SW 23rd Dr., Gainesville, FL 32608.*



Given these design criteria and the cost and compatibility constraints, the final specification sheet for our digital input and output (DIO) interface was as follows:

- Simple interface to a PC.
- Multiple digital input lines.
- Interrupt-on-input capability.
- At least one digital output, preferably more than one.
- Compatibility and availability across a wide range of platforms, including ISA, EISA, and MCA buses.
- Relatively low cost.

### Digital I/O Options

The PC architecture has a wide variety of input and output techniques available to it, from specialized DIO boards to the relatively crude game port. Each has some advantages and disadvantages for this type of system.

Specialized DIO boards are available for the PC, typically as 8- or 16-bit ISA boards with 48 I/O lines, configurable to generate interrupts on one of several different IRQs. While nearly ideal feature-wise, these boards are relatively expensive—from $40 or $50 to more than $1000. Even with the inexpensive boards, this cost becomes significant in high volumes. In applications using from 9 to 96 probes, EGPIC has been configured for use with these DIO boards. However, when only one to eight probes are needed, the DIO board is unnecessary. Also, since these boards require a free bus slot, some systems, such as laptops (a likely target platform), would be excluded from using EGPIC.

The PC's standard RS-232 serial port is suitable for this type of application, but the external EGPIC hardware would require an extra translation layer to generate RS-232-compatible bit streams from the eight digital inputs. This method is suitable for very large systems requiring hundreds or thousands of probes, but is unnecessarily complex for smaller-scale systems.

Among other deficiencies, neither the keyboard input nor the game port offer output capabilities, ruling out their consideration as suitable I/O interfaces for EGPIC.

This leaves the PC's printer parallel port. Like a dedicated DIO card, it offers digital output lines and interrupt-on-input capability (using either IRQ 5 or 7). Unlike DIO cards, it's available for all PC platforms and is relatively inexpensive. The parallel port's only shortcomings are that not all implementations have input capability and the port may already be in use by another device, likely a printer. However, many systems have two parallel ports; if not, a second parallel port is an inexpensive addition. And as for the lack of input capability, after some software tricks, a 100 percent compatible PC parallel port can, in fact, be used for up to 8 bits of digital input.

### Programming the Parallel Port

To illustrate the programming techniques discussed in this article, I've writ-

*(continued from page 64)*
ten the Parallel Port Digital Input Output (PPDIO) package, a rudimentary set of C functions that allow for reading and writing to the parallel port and installing an interrupt-service routine (ISR) to handle incoming data on the parallel port. Listing One (page 103) is PPDIO.H; Listing Two (page 103) is PPDIO.C.

The parallel port is programmed via three separate I/O registers: the input-only data register, the output-only status register, and the input/output control register. The addresses of these register ports differ depending on the machine, but they are usually offset from 0x378, 0x278, or 0x3BC. The base address for a particular LPT port is stored in the BIOS data area. The *PPDIO_GetLptAddress()* routine shows how to retrieve this information.

The data register (see Figure 1), located at the parallel port's base address, takes a standard bit mask that indicates which pins should be sent high and low. Sending information out the parallel port is accomplished with a simple OUT instruction. *PPDIO_SendByte()* handles this. The parallel port transmits this byte until told to transmit a different one, making digital output a trivial task. Note that while we can theoretically read the data register with an IN instruction, the byte read won't be incoming data—it will be the most recent data transmitted.

The data register can't be used for input, so we must use both the status and control registers; see Figures 2 and 3 for their respective layouts. Reading the status register is very straightforward, but keep in mind that the logic of pin 11 is inverted.

The control register is nominally an

| Bit | Pin/Function | Logic |
|---|---|---|
| 0 | 0 | 1=TRUE |
| 1 | 1 | 1=TRUE |
| 2 | 2 | 1=TRUE |
| 3 | 3 | 1=TRUE |
| 4 | 4 | 1=TRUE |
| 5 | 5 | 1=TRUE |
| 6 | 6 | 1=TRUE |
| 7 | 7 | 1=TRUE |

***Figure 1:*** *Data register (base address+0).*

| Bit | Pin/Function | Logic |
|---|---|---|
| 0 | (reserved) | — |
| 1 | (reserved) | — |
| 2 | (reserved) | — |
| 3 | 15 | 1=TRUE |
| 4 | 13 | 1=TRUE |
| 5 | 12 | 1=TRUE |
| 6 | 10 | 1=generate interrupt |
| 7 | 11 | 0=TRUE |

***Figure 2:*** *Status register (base address+1).*

| Bit | Pin/Function | Logic |
|---|---|---|
| 0 | 1 | 0=TRUE |
| 1 | 14 | 0=TRUE |
| 2 | 16 | 1=TRUE |
| 3 | 17 | 0=TRUE |
| 4 | IRQ enable | 1=enabled |
| 5 | (reserved) | — |
| 6 | (reserved) | — |
| 7 | (reserved) | — |

***Figure 3:*** *Control register (base address+2).*

*(continued from page 66)*
output-only register, but by taking advantage of the four output lines driven with open-collector drivers, we can force the control register into giving us input. If we produce a high TTL logic level at the control register's corresponding pins, we can drive the pins low via incoming signals. Thus, by setting the appropriate bits of the control register, the pins can be used as input. This is handled transparently when *PPDIO_InstallISR()* is called.

Reading the control and status registers is accomplished by an IN instruction at the port's base address and relevant offset. The routines *PPDIO_ReadStatusRaw()* and *PPDIO_ReadControlRaw()* illustrate how to accomplish this. Because several of the input lines have negative active logic placed upon them by the parallel port, helper functions that translate negative logic would be useful. The routines *PPDIO_ReadStatusCooked()* and *PPDIO_ReadControlCooked()* provide this functionality, along with converting reserved and unused bits to 0.

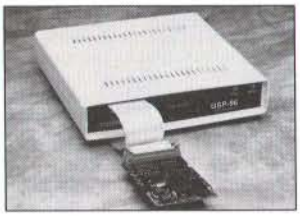
> *Like a dedicated DIO card, the PC's parallel port offers digital output lines and interrupt-on-input capability*

### Interrupt-driven Communications

Now that input and output have been addressed, all that's left is making the communications interrupt-driven. The parallel port's input lines could be polled; however, this would be cumbersome, time consuming, and error-prone. Having an interrupt generated whenever a digital line is sent high is a far more elegant means of input detection.

Assuming the target system supports it, interrupt-driven input on the parallel port is actually not very complicated to achieve. First, the control register must have its interrupt-enable bit set. Next, an ISR must be installed in the DOS interrupt vector table for the appropriate IRQ. Finally, the port's IRQ must be unmasked from the 8259 Programmable Interrupt Controller's interrupt-enable register. All of this is demonstrated in *PPDIO_InstallISR()*.

While programming in an interrupt-driven manner is theoretically simple, hardware support can be shaky. The printer port has traditionally utilized IRQ 7, but IRQ 5 isn't uncommon either. Even worse, some machines either have the parallel-port interrupt disabled altogether or have an alternate device (such as a network or sound card) using its IRQ. To compound matters, there's no easy way to detect which IRQ a given base address or LPT port corresponds to—this must either be known by the user or determined empirically by the program.









*(continued from page 68)*

To solve this problem, EGPIC uses a simple call-and-acknowledgment method of IRQ determination. This involves installing ISRs at IRQs 5 and 7, "calling" the EGPIC hardware (which acknowledges the call by generating an interrupt), then seeing which ISR is called.

> *By taking advantage of the four output lines driven with open-collector drivers, we can force the control register into giving us input*

If no ISR is called, either another IRQ is in use (doubtful, since the PC industry has fairly well standardized on IRQs 5 and 7 for the parallel port), or no IRQs are being used for the parallel port. It's simpler to have the user input which IRQ to use, but this demands a higher level of user knowledge than the application may reasonably assume. An interesting secondary use of this call-and-response procedure is for hardware testing—if a probe is known to be installed and fails to generate a response when requested, then that probe must be malfunctioning.

Interrupts are generated via pin 10, normally a printer's ACK line. A high-line level sent to pin 10 results in an interrupt being generated, assuming that all other relevant setup has been done.

During the development of EGPIC we found that both cable length (from the probe to the computer) and interrupt latency played a role in determining whether a signal actually existed at the inputs when the ISR was called. With long cable lengths and a fast computer, it was possible for some inputs not to be updated by the time the ISR was executed. Conversely, with a slow computer it was possible for the signal to have come and gone (depending on the length of the generated input) by the time the ISR was called. These timing problems can be compensated for in hardware, but not knowing of their existence can lead to some rather irksome bugs.

## Potential Problems

Two particularly bothersome problems came to light while developing the EGPIC system.

First, if the interrupt lines were allowed to float while the system was collecting data, the PC would likely lock up. This is because a floating line will often fluctuate between TTL TRUE and FALSE, causing thousands of interrupts to be generated every second, freezing up the system. Something as innocuous as accidentally pulling a cable loose or turning off the input hardware may cause a system lockup.

The second problem is that not all PC parallel ports are identical. Some parallel ports deviate considerably from the original PC's design, rendering this type of specialized input and output—which assumes 100 percent hardware compatibility—impossible. Unfortunately, only trial and error will determine which systems are nonstandard.

## Conclusion

Listing Three (page 103) is DIO.C, a program that implements simple interrupt-driven DIO with the parallel port. By itself, DIO.C is fairly useless—consider it more of a basic framework to draw upon than a real program. Any program that would use these routines would require some amount of custom hardware and software design.

At first glance, the antiquated design of the PC parallel port seems highly limited, but it can quite easily be configured as an inexpensive digital input/output interface. The hardware required is minimal: a one-shot circuit per channel and a single interrupt line for the logical OR of all the channels. The software, as demonstrated in this article, is quite simple and easily customized for specific applications. In our case, the parallel port satisfied all of our requirements superbly, enabling the EGPIC project to be both completed in a timely and cost-effective manner and distributed across a wide range of systems.

## Acknowledgments

We are grateful to Hok Chia and Sergey Kruss (University of Florida) for electronic technical assistance.

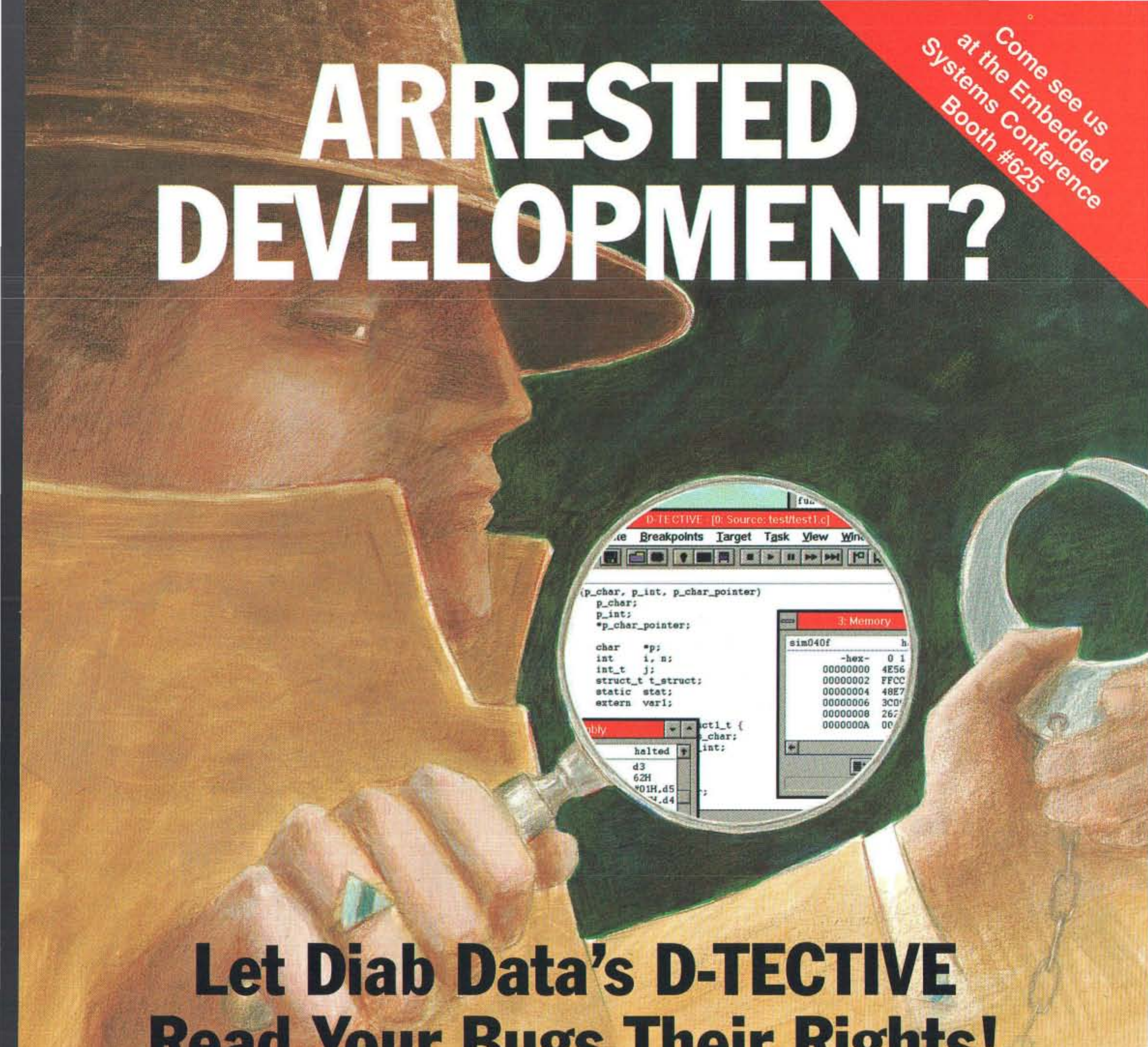
## References

Eggebrecht, Lew. *Interfacing to the IBM Personal Computer,* Second Edition. Carmel, IN: SAMS, 1992.

**DDJ**

**(Listings begin on page 103.)**

To vote for your favorite article, circle inquiry no. 7.




# ARRESTED DEVELOPMENT?

Come see us at the Embedded Systems Conference Booth #625

# Let Diab Data's D-TECTIVE Read Your Bugs Their Rights!

Are your efforts to get production-quality software out the door being dogged by suspicious, recurring, and persistent bugs? Put Diab Data's D-TECTIVE on the beat!

D-TECTIVE is the latest word in GUI, multitasking, multi-target, remote, source-level debuggers for Motorola's 680x0 and CPU32(+) family of embedded controllers and microprocessors, and it runs on the most popular platforms, in-circuit emulator tools, and real-time operating systems.

You've already heard about the legendary performance of Diab Data's D-CC/68K—the fastest, most reliable compilers for Motorola's 680x0 family of processors. Now, the industry's recognized leader in high-performance globally optimizing compilers has added proven debugging technology to its D-CC/68K software development tool arsenal.

Protect your code's performance and quality with D-CC/68K and D-TECTIVE—the smartest one-two punch available anywhere to get your Motorola-based embedded project to market fast and guarantee that it'll be a star performer.

Stop bugs and quality glitches from arresting your development! Let D-TECTIVE read'em their rights! Call Diab Data now to learn more about the most arresting software tools in the industry and our FREE evaluation program.

DIAB DATA
A Bull Company

U.S.: Telephone: (415) 571-1700 • Fax: (415) 571-9068
Europe: Telephone: +46-8 622-4422 • Fax: +46-8 622-4223

# EchoNets, E-memes, and Extended Realities

*Mobile computing requires new ways of thinking about networks*

Scott B. Guthery

The walkie-talkies of computing are personal digital assistants, or PDAs. Apple's Newton, AT&T's EO, Casio's Zoomer, and IBM's Simon all open the possibility of direct, computer-to-computer connections using wireless personal-communication systems. Network vendors will contend that central switches are necessary for communication among a large number of people, and while central switches do add features to network communication, switchless network communciation that relays messages from one node to another is also possible. This article will explore possibilities for switchless networks which I will call "EchoNets."

### EchoNets

Suppose that every minute or so your PDA broadcasts a message such as, "Curly here. Anybody out there?" Then, suppose I happen to walk by with my PDA turned on. It answers, "Yeah, Moe here. What's up, Curly?" Yours replies, "I've got 15 messages for you," and sends my PDA the messages. "Thanks," mine says, "and here are eight for you." "See ya, Curly," yours says. "Ciao, Moe," says mine, and they go their separate ways.

This is a basic EchoNet message exchange, a form of which is used in existing computer networks, including Usenet, FidoNet, and Relaynet. In fact, the news groups in FidoNet [Bush 93] are actually called "echoes," and mail is called "echomail." The difference between the use of the EchoNet-style protocols by PDAs and their use in existing networks is that the nodes in existing networks exchange messages with nodes that are known and relatively fixed over time. In a PDA EchoNet, a node is constantly polling for and talking to strangers.

Obviously, if I'd queued up a message to you in my PDA or if you'd entered a message to me in yours, we

would have communicated without a switch. This is plain-vanilla, peer-to-peer communication. But suppose my sister had written a message to you last night on her PDA. Sometime during the night, her PDA engaged in a message exchange with mine, and I unwittingly carried her message with me when I left for work this morning. When I walked by your PDA, I delivered her message to you. In a sense, my PDA was the network backbone that carried my sister's message to you.

EchoNets are by no means a recent discovery. One of the earliest networks in the Internet, the DARPA Packet Radio Network, used a "flooding" protocol, which is a form of EchoNet. And the gateways and bridges in modern switched networks are really nothing more than EchoNet nodes with fixed neighbors. So even switched networks may have EchoNet subnets.

However, we've become so accustomed to the features provided by switches that we think communication systems require them. While it will be useful for PDAs to be able to connect to pay-per-message switched networks such as cellular, telephone, satellite, and cable systems, it's important to realize that you are buying the added features of the switch and billing, as much as the raw network capability itself. It's also useful to realize that you can communicate without them.

### Receiver Addressing vs. Sender Addressing

Of course, one downside of EchoNet messaging is that the mail may not go through. And even if it eventually does go through, you have no idea about nor any control over how long it will take to deliver your message. It's like Usenet or FidoNet e-mail, only worse—much worse.

To get an EchoNet mail message from me to you, we need a chain of PDAs: First, I have to pass near A, then some time later, A has to exchange messages with B, and so forth, until you finally cross paths with Z. While it's helpful that the whole chain of PDAs need never exist in totality at any one point in time, you and I are clearly at the mercy of many chance events and random happenings. What this means is that you can probably get EchoNet mail reliably to people in the crowd you hang out with, but it's unlikely that you can get EchoNet mail to your friend in Tuva. In fact, it could be argued that EchoNets aren't good at all for person-to-person e-mail when you know exactly who you want to send the message to and what their address is.

Where EchoNets beat traditional e-mail (and telephone and surface mail,

*Scott is a scientific advisor at the Schlumberger Laboratory for Computer Science. He can be reached via Internet at guthery@austin.slcs.slb.com.*

```
To: EchoNet Implementers Everywhere
From: Earl of Echoes in Austin
Subject: Improved Short-Hop
                          Protocol
Keywords: EchoNet, Transfer Rules
Send-Date: September 6, 1993
Route: Bubba in Temple, Jenny Jet
                         in Dallas
```

***Figure 1:*** *An EchoNet header field.*

*(continued from page 72)*
for that matter) is when you don't know who you want to send the message to, or when you don't know how to get in touch with them. In this case, you want to broadcast your message in the hopes that the person or people you're looking for will receive it. Thus, it is the receiver or receivers of the message, rather than the sender, who determines to whom the message is addressed.

Just like a piece of e-mail, an EchoNet message comes with a number of header fields (see Figure 1) which describe it. Header fields help you scan the incoming EchoNet messages quickly to find the ones that are meant for you. You'll probably want to activate some sort of automatic text filter to sift through all the incoming messages and set aside ones that fit you and your profile of interests.

### E-memes

It's important to remember that, unlike Usenet newsgroup messages, you're both a potential recipient and a relay point for all of the EchoNet messages you receive. Just because you find a message you think is addressed to you, doesn't mean you shouldn't pass it on. There may be other people who are interested in the message, and you're part of the chain that is going to get it to them.

By default, you should also relay messages that aren't addressed to you along with ones that are. On the other hand, you're free to look over your message traffic and delete any messages you don't want your PDA to pass along. In an analogy to the memes (or thoughts) of human communication, EchoNet messages are kind of like e-memes. E-memes that people like—that they want to tell other people about—are passed on. E-memes that people don't like die off quickly, either by being read and deleted, or by being killed by automatic purge rules.

Besides passing e-memes, you can also annotate or elaborate on them. In this case, your observation becomes linked to the e-meme it comments on, and when the e-meme is sent to another PDA, these links are preserved so that you really can read and add to threads of thought.

*(continued from page 74)*

### Anonymity and Privacy

Have you noticed that Internet and FidoNet messages often arrive signed by everybody who handled them along the way? There are some interesting personal privacy implications if you extend this networking custom to an EchoNet. For example, if I get a message that has a path from Bob to Jim to Sally to Pete, then I could try to deduce that Jim was in the vicinity of Sally at some time. Due to name spoofing and path hacking (not to mention people borrowing each other's PDAs), this isn't ironclad evidence, but it is providing some information about the whereabouts of both your PDA and, by association, yourself.

Fortunately, EchoNet can forgo this cyberspace territorial-marking custom. There is nothing in the EchoNet relay algorithm that requires knowledge of how the message got to your PDA or the fact that it ever went through it. You can participate in EchoNets completely anonymously, or under a pen name. EchoNet pen names are like the handles of CB radio and the nicknames of Internet Relay Chat, with the advantage that you don't ever have to use an FCC-approved call sign or NIC-approved Internet address.

Privacy on an EchoNet is a little more problematic. If I can't understand what the message says, it's hard for me to figure out if it's for me or not. If it's encrypted and I don't have the key, then I'm pretty sure it isn't. Furthermore, if I can't read the message, then I can't determine if I want to pass it on or not and probably won't. Therefore, since encrypted e-memes will probably die out faster than unencrypted ones, I'd expect to see a resurgence of the clear text forms of encoding. This leads you to wonder which properties of an e-meme make it travel the farthest.

### An EchoNet Application: Measurements and Surveys

Psychologists who study cliques have devised fascinating ways of measuring the who-knows-whom connectivity between people. In one experiment, you're given a booklet describing a target person. This description does not include his or her name or whereabouts. You're asked to enter your name and address in the booklet, then pass it to somebody you know who stands a chance of getting the booklet to the target. The person to whom you give the booklet repeats the process, and sooner or later the booklet ends up in the hands of the person it describes. By counting the names in the booklet when it arrives, the psychologist obtains an upper-bound estimate of the who-knows-whom distance between you and the target. These experiments are called "studies of the small-world problem."

Suppose the people receiving the booklet had also been requested to enter some other information about themselves, rather than just entering their names and addresses. Then the booklets would accumulate a survey of all the people who handled them. Now suppose that the people are PDAs and the booklets are e-meme threads. What you have is a low-cost way of taking measurements and surveys.

For measurements, specially equipped PDAs can operate completely autonomously. At regular time intervals, the measurement's sensor is read, and a time- and location-stamped sensor value is queued as an outgoing e-meme. Over time, some of these readings find their way back to the person interested in them. While the coverage both in time and space is unpredictable, expenses are kept to a minimum and the flow is continuous.

An e-meme survey is more in the spirit of EchoNet. In this case, your PDA re-

# Subscriber Service

In order for **Dr. Dobb's Journal** to provide you with the best in Subscriber Service, we have compiled the listing below to help answer any of your service related questions. Please clip and save for easy reference.

### SUBSCRIPTION SERVICE

For all subscription inquiries regarding billing, renewal, or change of address: **Call Toll-Free** 1-800-456-1215 for U.S. and Canadian subscribers. **Call** 303-447-9330 for all other subscribers. Your mailing label will come in handy when speaking with our Customer Service Representatives.

To order new or gift subscriptions, please send your request to: Dr. Dobb's Journal, P.O. Box 56188, Boulder, CO 80322-6188.

### MOVING?

Please try to give us four to six weeks notice to ensure uninterrupted service. Subscriptions are not forwarded unless requested. Be sure to include your old address, your new address, and the date you'll be at the new address. Attach your mailing label showing your old address and account number - this is always helpful. Send to the above address.

### DUPLICATE COPIES?

Duplicated copies can occur when there is a slight variation in your name and address. Please send both mailing labels when notifying us of duplicates. Be sure to tell us which address you prefer.

### JUST MADE A PAYMENT – BUT STILL RECEIVING BILLING AND RENEWAL NOTICES?

A notice could have been generated just prior to your payment. If you have just made a payment, please ignore recent notices. It is most likely they have crossed in the mail.

### MAILING LISTS

From time to time we make our subscriber list available to carefully screened companies whose products may be of interest to you. If you would rather not receive such solicitations, simply send us your mailing label with a request to exclude your name.

*(continued from page 76)*
ceives a questionnaire as the head of a thread of responses. The thread head asks you to append the thread with your response to the questionnaire. You're free to throw the whole thing away or look at the responses of other people before you add your own. As with PDA measurements, we don't exactly have a controlled experiment, but then, we don't have to bear the cost of conducting one, either.

A downside of e-meme surveys can be getting the raw data back to the person conducting the survey. If the PDAs are moving around, you cover a wide area but may only get back a portion of the measurements you took. On the other hand, if the PDAs are relatively immobile, you'll cover less area but stand a better chance of collecting more data. In this case, you can take advantage of the fact that the PDAs are immobile and upgrade the basic EchoNet protocol to include a notion of routing.

What if, in sending around passive text fragments, there were some way of sending executable code fragments? I think this is what people have in mind when they talk about Knowbots and General Magic Telescript agents. If there were some way of telling friendly viruses from evil ones, then a code fragment that hopped from PDA to PDA, gathering up data and then heading home at the end of the day, would be a terrific way to cover a lot of territory quickly. If nothing else, the quitting-time algorithm will be fun to design.

### EchoNet Routing

What if our PDAs aren't roaming around but are sitting still, in a classroom, for example, or at a concert or in an office? Here, rather than trusting to random passoffs to get messages through, the PDAs can run a routing algorithm so that each PDA knows exactly which PDAs are out there and which PDAs a message has to go through to get to a particular PDA.

The simplest routing algorithm begins by each PDA figuring out who it is directly connected to. It does this using the usual message-exchange protocol, but rather than exchanging messages, it exchanges connectivity information. "Curly here. Anybody out there?" one says, and gets back a bunch of messages: "Yeah, Sleepy here. What's up?" "Yeah, Sneezy here. What's up?" "Yeah, Doc here. What's up?" "Yeah, Bashful here. What's up?" Now Curly knows he's directly connected to Sleepy, Sneezy, Doc, and Bashful, and each of these knows they are directly connected to Curly.

With this information, Curly can address messages directly to particular PDAs rather than broadcasting them to everybody and having to deal with all their responses. "Curly here. Who are you connected to, Sleepy?" Sleepy responds to Curly, "Sleepy here. I'm connected to Doc and Snow White, Curly." "Ahah! A new player," thinks Curly. What Curly has discovered is that there is a Snow White out there that he can get a message to by way of Sleepy. "Curly here. Give this to Snow White, Sleepy: 'Yo, SW, what's cookin'?' "

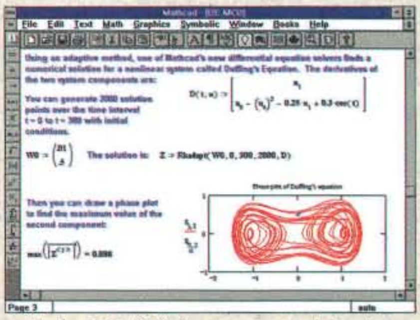
What you have here is explicit routing. Rather than just broadcasting a message into the ether, Curly sent it directly to Sleepy along with explicit instructions to pass it directly to Snow White. We've also added the peer-to-peer "Who are you connected to?" message. By coupling this message with direct addressing, any PDA can discover the entire known universe and its connectivity. Knowing this, your PDA can present you with a list of all the PDAs with which you can communicate on EchoNet and can send any message directly to the one you pick. In a sense,

# WE'VE EXAGGERATED HOW MUCH CALCULATING POWER IS IN NEW MATHCAD PLUS 5.0. BUT ONLY SLIGHTLY.


It gives you more advanced math capability than ever before. It lets you tackle harder problems and solve even tougher equations. In short, it's the most powerful, most advanced version of Mathcad® ever released.

And that's no exaggeration. More powerful than spreadsheets or calculators, easier than programming languages, new Mathcad PLUS 5.0 gives engineers, scientists and educators more tools to do calculations with greater speed and ease.

You get more functionality for computing derivatives and integrals, differential equations, advanced vector and matrix operations, statistical functions, curve fitting, and fast Fourier and wavelet transforms. You can choose from a wider range of symbolic capabilities, and graph in 2-D and 3-D polar, contour and parametric plots. MathSoft's Electronic Books, based on the most popular reference books, let you instantly cut and paste hundreds of formulas into your work. And with Mathcad PLUS Function Packs you can add even more remarkable calculating power in specific disciplines like signal processing, data analysis, statistics and graphics.



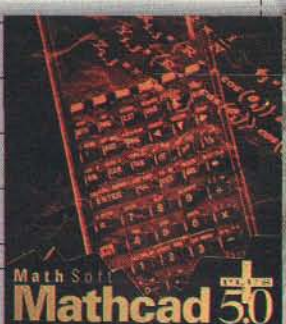

*Mathcad PLUS 5.0's on-screen worksheet lets you easily combine equations, text and graphics.*

Plus, like its Mathcad predecessors, it's as easy and intuitive as using a scratchpad. Simply enter equations in real math notation anywhere on Mathcad PLUS 5.0's on-screen worksheet. Add text and graphics. Change variables and instantly update your work. Mathcad PLUS 5.0 calculates answers quickly and accurately, then prints your results in impressive, presentation-quality documents.

So try Mathcad PLUS 5.0 today, and tomorrow you'll be ten times more productive. Well, maybe we're exaggerating again. But only slightly. Mathcad PLUS 5.0 is priced at just $299.95*. To order, or to receive even more information, call **1-800-967-5075**. Or mail or fax the coupon below (Fax: 716-873-0906).

**FREE MATHCAD PLUS 5.0 INFORMATION KIT**
For more information on Mathcad PLUS 5.0, mail or fax this coupon.

Name ______ Title ______ a22a2
Company ______
Address ______
City ______
State ______ Zip ______
Country ______ Phone ______




MathSoft, Inc. P.O. Box 1018, Cambridge, MA 02142-1519 USA • Phone: 1-800-967-5075 • Fax: 716-873-0906 • MathSoft Europe, P.O. Box 58, Livingston, UK EH54 7AE Phone: +44.506.460373 • Fax: +44.506.460374 

**CIRCLE NO. 900 ON READER SERVICE CARD**

*(continued from page 78)*
your PDA is functioning as a router or a switch as well as a source, sink, and passive relay point. You're realizing some of the advantages of a switched network without building a central switch through which all traffic must flow.

There are hundreds of network-discovery and network-routing algorithms and protocols. Many can be used in EchoNets and switched nets. In fact, an EchoNet can be thought of as just a network, in which every node is also a router. Recently, the Internet technical community has become interested in supporting Internet connectivity to mobile hosts (see the accompanying text box entitled "Mobile Internetworking") and has published the "Internet Packet Transmission Protocol," which discusses a possible routing algorithm for this situation.

### EchoNets as Distributed Systems

This primitive network-discovery and routing algorithm is an example of a large class of distributed-system algorithms that has received attention over the last 15 years. Dijkstra's classic paper [Dijkstra 80] and Chang's independent discovery [Chang 82] have set the tone and direction for much of this work. Chang's paper is a more readable introduction to distributed algorithms even though Dijkstra's paper has priority. Yang and Marsland's recent note [Yang 93] is an excellent annotated bibliography on two important problems in this field.

Dijkstra and Chang showed that it is possible to design practical algorithms for EchoNets which enable any node in the network to discover information about the whole network or about any particular node in the network. In fact, Chang called these algorithms "echo algorithms" because a broadcast question produces an echoed response. "Practical" here means that the answer is obtained in a deterministic and computable amount of time and that the EchoNet message traffic generated by the broadcast request eventually dies out.

The Dijkstra/Chang echo algorithm proceeds as follows: The initial, inquisitive node sends its question to each of the nodes to which it is connected. Upon initially receiving the question, each node relays the question to all the nodes to which it is connected except for the initial node. If the receiving node has no other nodes to which it can send the question, then it sends the accumlated answer back to the node that sent it the question. Finally, when a node receives answers from all the nodes to which it sent the question, it in turn sends the accumulated answers to the node that first sent it the question.

In his paper, Chang gives a number of applications of this basic echo algorithm along with some performance calculations and special-case improvements. One of the applications, the Single-Source Sort, is particularly applicable to our PDA-to-PDA communication situation. The idea is that a new node is joining an existing EchoNet and wants to pick a unique identity. The new node sends out the question, "What is your name?" Each node's echo is its own name, appended to the list of names it has received from the nodes it has contacted. What arrives back at the new node is a list of the names of all the nodes in the EchoNet. All the new kid on the block has to do now is pick a name that isn't on the list.

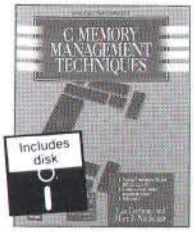
### Global State and Cooperative Behavior

Dijkstra/Chang-style echo algorithms are fine for determining static properties of an EchoNet (such as the list of the names of all the nodes in the network); but what about dynamic properties? Suppose all the nodes in an EchoNet

# 4 books for only $4.95

**Values to $183.00**

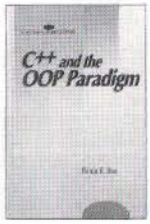
## when you join the *Computer Professionals' Book Society*




4191P $32.95

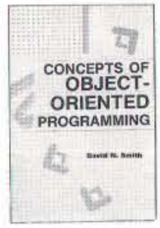

9309P $29.95


881933P-XX $39.95
Counts as 2

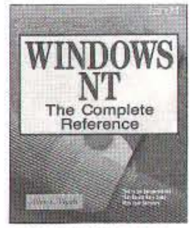



051140H $40.00
Hardcover

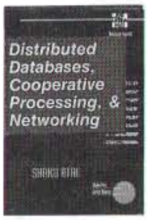

016732H $40.00
Hardcover



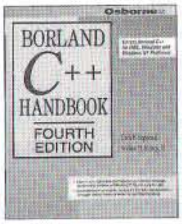

10039P $24.95



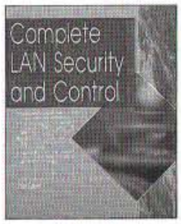

881832XP-XX $29.95
Counts as 2



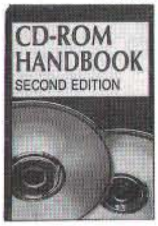

157673H $50.00
Hardcover




8819601P-XX $34.95
Counts as 2

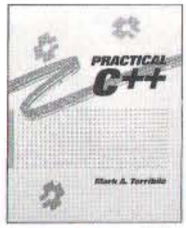



4490H $39.95
Hardcover

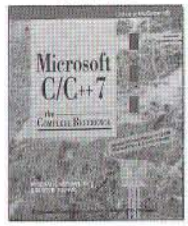



0566933H-XX $70.50
Counts as 2/Hardcover

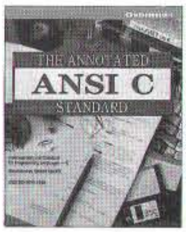

3484H $34.95
Hardcover



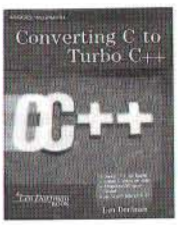

0637385H-XX $39.95
Counts as 2/Hardcover



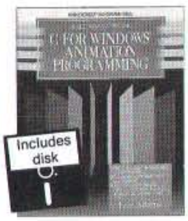

881664P-XX $29.95
Counts as 2




8819520P $39.95




4084H $39.95
Hardcover

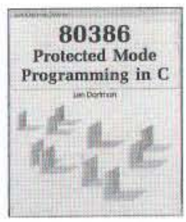



4114P-XX $39.95
Counts as 2


4340P $34.95


3049P-XX $29.95
Counts as 2

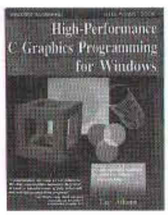

881796P $27.95



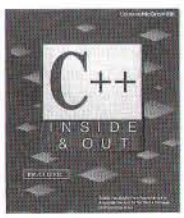

3736P $24.95


006215H $40.00
Hardcover

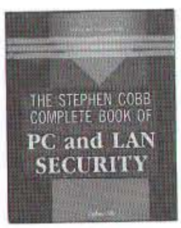

881653P-XX $29.95
Counts as 2




4103P $24.95




881809P-XX $29.95
Counts as 2

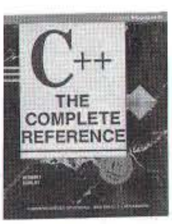



3280H $36.95
Hardcover


005076H-XX $50.00
Counts as 2/Hardcover

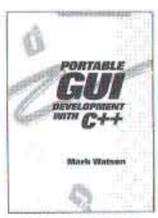

881691P-XX $39.95
Counts as 2


4341P $34.95

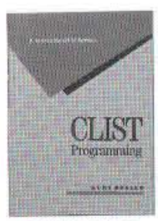



881654P $29.95

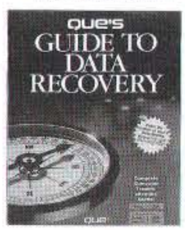

586118H-XX $39.95
Counts as 2/Hardcover




0684898H $45.00
Hardcover

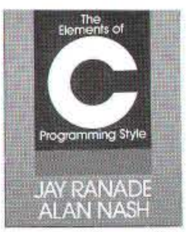



006551H $49.00
Hardcover



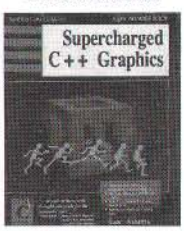

586104P $29.95


051216H-XX $39.95
Counts as 2/Hardcover




051278P $29.95


4196H $32.95
Hardcover




3489P $24.95

All books are softcover unless otherwise noted. Publishers' prices shown. If you select a book that counts as 2 choices, write the book number in one box and XX in the next. A shipping/handling charge & sales tax will be added to all orders. ©1994 CPBS

If card is missing, write to:
Computer Professionals' Book Society, Blue Ridge Summit, PA 17294-0870

### As a member of the Computer Professionals' Book Society . . .

you'll enjoy receiving Society bulletins every 3-4 weeks containing exciting offers on the latest books in the field at savings of up to 50% off of regular publishers' prices. If you want the Main Selection do nothing and it will be shipped automatically. If you want another book, or no book at all, simply return the reply form to us by the date specified. You'll have at least 10 days to decide and if you ever receive a book you don't want, due to late mail delivery of the News, you can return it at our expense. And you'll be eligible for *FREE BOOKS* through the Bonus Book Program. Your only obligation is to purchase 3 more books during the next 2 years, after which you may cancel your membership at any time.

DDJ494

*(continued from page 80)*
wanted to cooperate in accomplishing a task of some sort. How would they keep track of the current state and progress of their combined effort, or stay coordinated?

One way would be to have everybody synchronize their actions to a global clock, then treat the dynamic state as simply a series of static states separated by network-wide time synchronization points. From an individual node's point of view, the drill might look something like this (see, for example, [Flammer 92]):

1. Do something useful.
2. Wait until the global-synchronization point.
3. Exchange what you've done with everybody else and find out what everybody else has done using an echo algorithm.
4. Figure out what to do next.
5. Go to step #1.

While there are a number of global-clock and virtual-time algorithms which can be used for the global-synchronization point [Yang 93], you get the feeling there's an excessive amount of overhead in this approach—theremust be a less West Point and more Mill Valley way of achieving cooperation.

Chandy and Lamport [Chandy 85] describe an algorithm whereby nodes in an EchoNet can determine the global state of a cooperative effort without a global clock. They called their algorithm a "distributed snapshot" by analogy to a group of photographers (the nodes) who take several pictures (the local states) and piece together the results (by exchanging messages) to form a "meaningful" panoramic picture (the global state) that is larger than a picture that any one photographer's camera could handle. In this context, "meaningful" means that the composite picture is sufficient for coordinating the nodes and getting the cooperative effort accomplished.

The Chandy/Lamport algorithm provides a method for "strobing" the recording of a node's local state by a mechanism other than a global alarm clock. The method is based on the sending of a special "Record your state!" message around the network. After the message has been received and obeyed by all nodes, the recorded local states are collected to form a description of the global state; this global state description is distributed to all nodes using an echo algorithm.

Since the "Record your state!" message reaches nodes at different times, you have to record not only the state of each of the nodes but the state of what causes nodes to change state; videlicet, the in-transit message traffic between nodes. The resulting global state is rather like a little film clip that we can run forward and backward to see what the network was up to during a tiny interval of time. The Chandy/Lamport algorithm is a careful specification of how the local states of the nodes and the communication channels are to be recorded so that this movie is a useful representation of the net's state.

The Chandy/Lamport distributed snapshot algorithm works like this:

**Initiation Rule:** Send the "Record your state!" message to each node to which you're connected and then record your state before you send any further messages.

**Unrecorded State Rule:** If you receive a "Record your state!" message and you have not already recorded your state, then: 1. Record your state; 2. record the fact that the state of the channel between you and the node that sent you the message is "Empty"; 3. start recording all subsequent messages you receive from other nodes; and 4. send the "Record your state!" to each node to which you are connected.

**Recorded State Rule:** If you receive a "Record your state!" message and you already have recorded your state, then record the fact that the state of the channel between you and the node that sent you the message is the sequence of the messages you got from this node between the time you recorded your state and the current "Record your state!" message.

The algorithm gives receivers the obligation of recording the in-flight messages. The recording starts at a node

# Mobile Internetworking

A number of similar schemes (Ioannidis, Uehara, and Wada, for example) have been proposed for extending TCP/IP, and hence, the Internet, to mobile computers. The situation is a little more complicated because, as originally conceived, TCP/IP addresses combine two distinctly different pieces of information: unique name and current location (kind of like "Minnesota Fats" or "Boston Blackie"). When host computers were immobile, this didn't matter; if they did move, we changed their names. Clearly this solution won't work for a computer tooling down Route 66.

A design criterion for all of the mobile-internetworking proposals is to minimize the impact of supporting mobile hosts on the existing network as much as possible. Thus, not only should everything that works today continue working, but a stationary host should be able to communicate with a mobile host just as if it were another stationary host. Basically, this means that the mobile host's address doesn't change, at least from the point of view of other hosts communicating with it.

The Internet Packet Transmission Protocol (IPTP) proposed in a July 1993 Internet Draft (Wada) endows a mobile host with two addresses: a home address (the unique name, "Blackie") and an away address (the current location, "Boston"). The home address doesn't change and is the permanent name of the host known to the world. The away address does change and is the address at which the mobile host can currently be reached. The only change to the network is the addition of a piece of software called a Packet Forwarding Server to the mobile host's home network, which keeps track of where the mobile host is and forwards messages to it.

As the mobile host moves around, it acquires an away address from each local network whose territory it enters, and it sends this away address back to its home network's Packet Forwarding Server. In this way the home network always knows where the mobile host is and how to reach it. Messages to a mobile host are always sent to its home address; when they're received, the Packet Forwarding Server readdresses them to the mobile host's away address. Messages from a mobile host go directly to whom they are addressed and need not detour through the Packet Forwarding Server on the mobile host's home network. The return address on these messages is the mobile host's home address, not its temporary away address.

As hosts and routers on the Internet are willing to become more mobile-host-aware, there are a number of efficiencies that can be introduced into this minimum-impact protocol, and many of these are described in the referenced papers. For example, a sender might indicate that it is willing to track the mobile host as well so the mobile host could set its return address to its away address rather than its home address.

—S.B.G.

*(continued from page 82)*
when the first "Record your state!" message appears on any channel and stops channel-by-channel when "Record your state!" appears on each channel.

**Extended Realities**

What the Dijkstra/Chang and Chandy/Lamport algorithms give us is a way for many mobile computers to act cooperatively. While each individual PDA is keenly aware of its own surroundings, it can also count on the "eyes and ears" of the other PDAs in its EchoNet to act as lookouts in regions beyond its own ken. I think of PDAs knitted together by these algorithms as being similar to the compound eye of an insect or a very large array radio antenna. In a sense, the reality of each PDA has been extended to the area covered by the entire EchoNet of which it is a member.

It's interesting that this relatively complex form of network behavior has been achieved without a central switch. Switchless networks like EchoNets certainly have their drawbacks, such as indeterminate message delivery. The advantages, however, include robustness due to absence of a single point of failure and the ease with which nodes can enter and leave the communication mesh. In the era of mobile wireless computing, we may find situations where it just doesn't make sense to send the message downtown and back if it only has to get to someone standing next to me. We may also find useful forms of network communication that don't send us a bill at the end of the month.

**Bibliography**


Acharya, Arup, and B.R. Badrinath. "Delivering Multicast Messages in Networks with Mobile Hosts." *Proceedings of the 13th International Conference on Distributed Computing Systems.* May 25–28, 1993, Pittsburgh, PA.

Bush, Randy. "FidoNet: Technology, Tools, and History." *Communications of the ACM* (August 1993).

Chandy, K. Mani, and Leslie Lamport. "Distributed Snapshots: Determining Global States of Distributed Systems." *ACM Transactions on Computer Systems,* (February 1985).

Chang, Ernest J.H. "Echo Algorithms: Depth Parallel Operations on General Graphs." *IEEE Transactions on Software Engineering* (July 1982).

Chow, Ching-Hua. "On Multicast Path Finding Algorithms." *Proceedings of IEEE Infocom '91.*

Dijkstra, E.W., and C.S. Scholten. "Termination detection for diffusing computations." *Inf. Proc. Lett.* (August 1980).

Ioannidis, John, and Gerald Q. Maguire, Jr. "The Design and Implementation of a Mobile Internetworking Architecture." *Proceedings of the 1993 Winter USENIX Meeting,* January 25–29, 1993, San Diego, CA.

Flammer, George H. *Method for Synchronizing a Wide Area Network without Global Synchronization.* U.S. Patent #05130987.

Ioannidis, John. *Protocols for Mobile Internetworking.* Ph.D. Thesis, Columbia University, 1993.

Trask, Jeremy R., and Anthony Wiener. "Data Communication System." U.S. Patent 4,937,569, June 26, 1990.

Uehara, Keisuke, et al. "Enhancement of VIP and Its Evaluation." *Proceedings of INET '93,* August 17–20, San Francisco, CA.

Wada, Hiromi, Tatsuya Ohnishi, and Brian Marsh. "Packet Forwarding for Mobile Hosts." Internet Draft, July 1993.

Yang, Zhonghua, and T. Anthony Marsland. "Annotated Bibliography on Global States and Times in Distributed Systems." *ACM Operating Systems Review* (July 1993).


**DDJ**

To vote for your favorite article, circle inquiry no. 8.

DESQview/X™

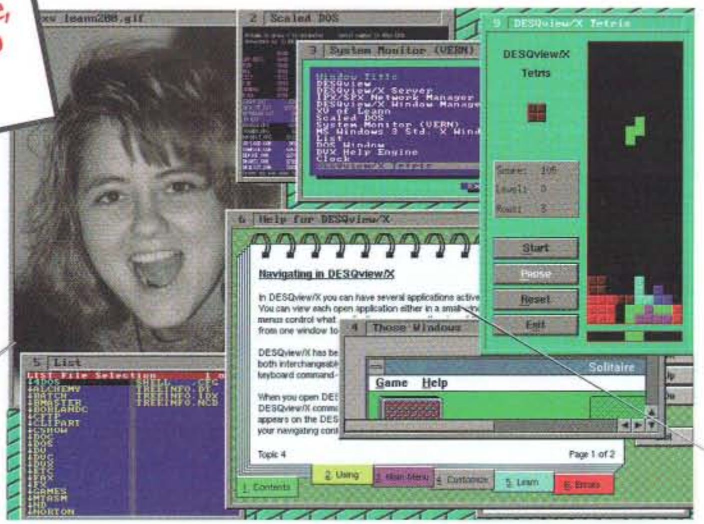
# The simplest way to integrate DOS, MS Windows and X Window software on your PC—or any other PC on a network.

**New Version 1.2** includes enhanced network support: PC-NFS, Hewlett-Packard LAN Manager, Microsoft LAN Manager, Beame & Whiteside, Wollongong—in addition to PC/TCP and Novell LAN Workplace for DOS

DOS and Windows applications compatibility.

Proven memory management and diagnostics with both Manifest and QEMM.

DESQview/X has EGA, VGA, 8514A, Super VGA and DGIS high resolution graphics support.



Comfortable 'look and feel' for both keyboard and mouse users.

With scalable window capability provided by Adobe Type Manager,™ you can get more DOS windows on the screen than ever before.

Any DESQview/X window can be made into an icon at any time.

Online help is two keystrokes away.

## Multitasking Power

DESQview is the recognized pioneer in DOS multitasking and is at the very heart of DESQview/X—giving you proven high performance multitasking of DOS and Microsoft Windows programs side-by-side.

## X Window System Graphics

The X Window System gives DESQview/X its graphical interface And its ability to access DOS text and MS Windows programs running remotely on your network. Add the optional TCP/IP Network Manager and it gives you the ability to access remote X Window programs on workstations such as an IBM RS/6000 or a SUN SPARCstation. Or vice versa.

## Adobe Type Manager

The accepted standard in scalable font technology is built in so that X Window System programs can use scalable fonts. And it also gives you the ability to view DOS text programs in scaled windows.

## Program Memory Management

DESQview/X has the memory management services of advanced operating systems—thanks to the built-in Rational Systems DOS4GX DOS extender (16 and 32 bit), shared libraries, dynamic link libraries (DLLs) and virtual memory.

## Superior DOS Memory Management

DESQview/X comes with our award-winning, best-selling expanded memory manager, QEMM, and the highly acclaimed Quarterdeck Manifest to ensure that your PC's memory is always at its optimum and that your DOS programs have the absolute maximum memory available.

## Customization

With DESQview/X's customizable menus, graphical desktop, keystroke macros, mark and transfer and online help, it is very easy to tailor the your own PC to your exact working needs. In addition, you can choose from two optional 'look and feel' packages: OSF/Motif and OPEN LOOK.

## Includes Valuable Companion Programs

DESQview/X comes with three companion programs. Application Manager for launching and organizing programs. File Manager for managing local and remote files. And Icon Editor for creating icons.

Quarterdeck makes your PC a better place to work, whatever software you prefer to use.

**Yes! Please Send:**

| Qty | Product | 5-1/4 | 3-1/2 | Each | Totals |
|---|---|---|---|---|---|
| | DESQview/X | | | $275.00 | |
| | DESQview/X Network Mgr. (TCP/IP) | | | $200.00 | |
| | DESQview (for 286 PCs) | | | $99.95 | |
| | DESQview 386 (for 386 and 486 PCs) | | | $149.95 | |
| | Shipping & Handling $10 (Canada & USA only) | | | | |
| | California Residents add 8.25% | | | | |
| | Grand Total | | | | |

Payment method ☐Visa ☐MasterCard Expires ____/____

Card # ____________

Call (800) 354-3222 ext. 1D1 for fastest service

Please allow 3 weeks for delivery

Name ____________

Address ____________

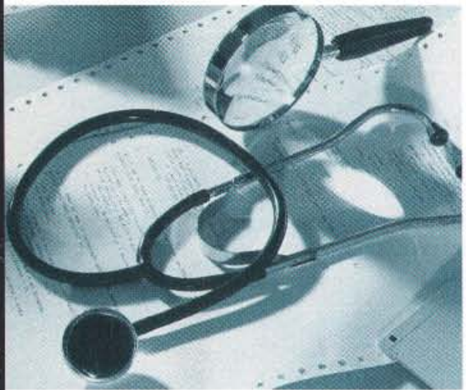
City ____________ State ____ Zip ____

**Quarterdeck**

Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405 (310) 392-9851 Fax (310) 314-4219
Quarterdeck International Ltd., B.I.M. House, Crofton Terrace, Dun Laoghaire Co. Dublin, Ireland Tel.(353) (1) 284-1444 Fax: (353) (1) 284-4380



CIRCLE NO. 501 ON READER SERVICE CARD



# Help for Windows Help Authors

*Windows help authoring tools provide quick relief*

Al Stevens

To be taken seriously, a Windows application must provide online help. Users have come to expect it, and developers have little choice but to provide it. However, like staff meetings, program documentation, and user's guides, it's a task that programmers approach as willingly as they would a root canal. But like it or not, most Windows developers must eventually build a help database. Fortunately, Windows includes WINHELP.EXE, an application that displays online, context-sensitive help in a standard format. You design a help database and build the hooks into the application. WINHELP does the rest.

Building a help database, however, is no easy task. If you're lucky, your boss hires a professional tech writer to do most of it. There is more to the job than writing the words and composing the pictures. You use a number of unrelated tools to convert the help words and graphics into a database format that WINHELP recognizes. You can work with these tools in their native autonomous environments or use a Help authoring tool to integrate them into a project-oriented toolset. This article describes the components of a Windows help database, addresses the manual procedures for building one, and discusses three Help authoring tools that ease the process. One tool, the Windows Help Author, comes as an unsupported application on the Microsoft Developer Network (MSDN) CD. The other two, Windows Help Magician from Software Interphase and RoboHelp from Blue Sky Software, are commercial tools from third-party vendors.

### About WINHELP

WINHELP is an independent Windows application that comes with Windows. Developers use it to provide online help similar in look and feel to that of other Windows applications. WINHELP displays help text and graphics from databases that conform to a prescribed format. The format supports hypertext links, keyword searches, graphical displays and controls, and navigational controls. Applications programs associate their run-time contexts with specific topics in the Help database. The developer composes the help database, assigns run-time context identifiers to the help topics, and puts the associated context-sensitive hooks in the applications code.

A help database can have text, graphics, motion video, and sound. The text can include highlighted phrases that you click on to pop up informational windows or jump to other topics in the text. Graphical elements such as tool-bar buttons, icons, and screen shots can be displayed and clicked on. The help document can talk, display pictures, play music, and show movie clips. There is an automatic table of contents and a keyword search feature. There are navigation functions that jump forward and backward through the topics. The user can place and retrieve bookmarks in the text. All of these features are implemented by WINHELP based on a database that the developer builds.

Although WINHELP typically provides online help to applications, its hypertext, multimedia, and navigational features make it useful for presenting other kinds of information. You can run WINHELP either from within an application, or as a stand-alone program and command it to display text and graphics from any conforming database. WINHELP is commonly used for online reference and users' documentation for compilers and other applications. When you see a Program Manager group with one or more prominent yellow question-mark icons, there is a good chance that they each run WINHELP to display a different documentation database. There are even "readme" files implemented as help databases.

### Help-Project Tools and Components

Building a help database involves several tools. You need a word processor that reads and writes the so-called "rich-text format" (RTF), which is an abominable concoction of embedded ASCII tokens that define how a document should appear. The RTF editor is used to compose the help database. Theoretically, you could use an ASCII text editor to build an RTF document, but it's not advisable. You're better off using a word processor that works with the format and displays comprehensible text on the screen.

A second text editor, such as Notepad, that works with ASCII text maintains the help project file, which describes various components in the database. If your database includes graphics, you need a program to produce bitmap files. If the user is to make selections by clicking on parts of the graphics, you need the Hot-Shot Editor. The Help Compiler reads the project, text, and bitmap files and compiles the help database into the format that WINHELP expects. (The Help Compiler and Hot-Shot Editor are bundled with most Windows software-development environments and are included in the SDK.) Finally, you need your own software-development environment to put help context identifiers into the application.

Among the project components you'll use when building a help database are:

**Help Project File.** A help database uses an ASCII text project file with the .HPJ filename extension. The project file contains options for the Help Compiler, including the name of the .RTF text files, and the title and size of the help window. It also associates string topic iden-

*Al is a* DDJ *contributing editor. He can be reached through the* DDJ *offices or on CompuServe at 71101,1262.*

tifiers that you place in the help text with integer context identifiers in the programs.

**Help Text.** The help text contains the narrative text for the help database, tokens that specify the filenames and position for graphics, topic names and identifiers, and the linkages for hypertext references and keyword associations.

The RTF word processor must be able to encode double-underlined and hidden text and insert footnotes into the document, all by using RTF protocols. Footnotes, which are tagged with $, #, and K characters and appear just ahead of the title for each topic, provide mnemonic identifiers for the topic, titles for the table of contents, and keywords for the search. Underlined text identifies hypertext phrases. Hidden text immediately follows the hypertext phrases and specifies the link's mnemonic identifier. Word for Windows 2.0 and Ami Professional both have these capabilities.

**Multimedia.** A help database can include graphics, sound bites, and movie clips. You include these elements by putting tokens in the text that identify what they are and their filenames. For example, to insert a bitmap you put the following token into the text at the character position where you want the upper-left corner of the bitmap to show:

```
{bmc dolly.bmp}
```

The *bmc* keyword specifies a bitmap. *dolly.bmp* is the name of the file in this example that contains the bitmap. The Help Compiler uses the token to build the graphic rendering into the database.

If the user clicks on a graphic to jump to another topic, you underline the token and add a hidden topic identifier to link to the topic. If parts of the same graphic point to different jump links, you use the Hot-Shot Editor to identify the coordinates of each jump area. For example, I put a picture of the application's tool bar in the help database and used each tool button to jump to the topic that describes its function. A weakness of the help development system is that graphics in the database slow the Help Compiler down to a crawl when you use the compression options to build the database. A big help file with many large pictures can take hours to compile.

### Context-Sensitive Help

Most help databases support context-sensitive help. If they do not, the user must start at a table of contents or do a keyword search to find a particular help topic. By associating help topics in the database with menu selections, dialog boxes, controls, and other application-specific contexts, you give the user the ability to go directly to the help topic that discusses the currently selected application context. For example, suppose that you assign context identifier 3 to the EDIT_MENU identifier. These are arbitrary values that you decide to use. In your program, you associate the integer 3 with the Edit menu label on the menu bar. In the help text, you associate the EDIT_MENU identifier with the topic in the text that explains the menu. When the user selects that menu and asks for help, winhelp.exe displays the associated topic.

To integrate a help database with an application, you modify the source code to specify the database name and to associate context identifiers with different parts of the application. How you do this depends on the programming language. Visual Basic's Options/Project menu opens the Project Options dialog box where you add the name of the help database. The Menu Design Window includes a *HelpContextID* field for the numeric value associated with the associated help topic. The Properties windows for the application's controls include similar *HelpContextID* fields.

A C program that uses the Windows SDK API intercepts the WM_KEYDOWN message and watches for the F1 key, either from the application window's processing module or by using the *SetWindowsHookEx* function to install a filter function that intercepts messages to dialog boxes. Once the key is pressed, the program calls the SDK's WinHelp function passing the name of the help database and the context identifier number.

Microsoft Foundation Classes programmers associate help contexts with controls using the MAKEHM tool, which constructs topic mnemonics from the source-code control identifiers and assigns context identifiers to them. The message map associates the Windows ID_HELP message to *CWinApp::OnHelp*. AppWizard builds this framework automatically when you elect to include context-sensitive help in your application.

### Building a Database: The Manual Approach

The Windows Program Manager has features that help organize the tools into what approaches an integrated fashion. Recently I worked on a help database for a Visual Basic application. I wanted VB and the help-authoring tools available at the same time so that I could put context identifiers in the program while I wrote topics in the help database.



I set up a Program Manager group for the project. An icon runs Visual Basic with the application's makefile on the command line. That takes care of the software-development side of the project. Another icon runs Notepad to edit the help project file. A Word for Windows icon starts Word to edit the RTF text file. An MS-DOS prompt icon starts a DOS batch file that runs the Help Compiler. I used Paintbrush to build and change bitmaps; it has an icon in the Program Manager group. Finally, an icon runs WINHELP itself to view the help document during each stage of its development. These tools sit together as Program Items in a Program Group with the startup subdirectories and command-line document files built into their properties. Thus, not only do I avoid rummaging through all of the groups to find and run them, but they start up with the help files loaded and ready to modify.

The manual approach works, but it is not perfect. Getting into and out of Word involves telling Word each time to convert the RTF format. The help document in Word does not resemble the display that WINHELP uses. You get neither a visual tool nor WYSIWYG. To see the real thing, you must run the Help Compiler and compile the whole database, which can take a long time. Inserting the correct footnote tokens with the correct footnote values is a tedious and error-prone process. Remember that you are using the features of a word processor to create links and chains in a textual database, a text editor to associate the link identifiers with numbers in the project file, and a software-development environment to put the numbers in the source code. There are no built-in integrity checks. Nothing ensures that you properly coordinate the contexts and topic identifiers among the three files. Some, but not all, of these problems go away when you use a help-authoring tool.

### Microsoft Windows Help Author

The Microsoft Developer Network CD contains an "Unsupported Tools and Utilities" section that includes a program named "Help Author," which is easy to use and well documented. The MSDN CD-ROM includes as a bonus an extensive Help Authoring Guide that covers the creative side of the job.

Help Author has two parts: an application named "Help Project Editor" and a Word for Windows template. The Help Project Editor uses dialog boxes to collect the information for the ASCII project file. You don't have to deal with that file again. It also automates the interface with Word, launching it with the template installed and the RTF file loaded.

A help database can contain more than one RTF file, and Help Project Editor keeps them in a list. You can also launch the Help Compiler and WINHELP to view the currently compiled database.

The Word template adds three tool buttons to the Word tool bar. The first one opens a dialog box that lets you change the footnote values in the current topic. You can add, change, and delete the topic's title, context mnemonic string, keywords, browse sequence, and so on, all without having to deal with Word's footnote commands. The second tool button opens a dialog box that lets you insert jump and pop-up links into the database, automatically applying the underlined and hidden text attributes. If you select a phrase that is already in the text, the operation uses it. Otherwise it inserts whatever you put in the dialog box as the link phrase.

The third tool button compiles only the current topic into a temporary help database and calls WINHELP to display it. You preview a topic — text, graphics, sound, and movies—exactly as the user sees it and without recompiling the entire database. What you see in Word and what WINHELP displays is usually quite different. You need to view your progress in small increments, and this feature supports that need. It is Help Author's strongest advantage over the other tools, and anyone who develops a sizable help database wants this capability. The other tools do not have it.

Help Author does not integrate graphics and multimedia tools. You still have to launch them yourself and write their files into the proper subdirectory so that the Help Compiler finds them.

An amusing side to Help Author is that its own help database has context errors. Most of the Help buttons on its dialog boxes link to help topics that do not exist, although there are topics in the database to cover the functions of the dialog boxes. Nonetheless, Help Author smooths several of the wrinkles out of the manual procedure, automates most of the tedium of using Word to build the database, and is well worth trying. As a Windows developer, you should have the MSDN CD-ROM, anyway. Help Author is a bonus.

### Windows Help Magician

Windows Help Magician from Software Interphase has some good features, some bugs, and some annoying quirks. Among its quirks is that the setup includes a package called "Bitmap Magician." Its purpose is to let you build a pseudofont by converting the characters in an existing font into bitmaps that you can include in the help database. Help Compiler uses only a few fonts and doesn't accept all of the characters in the fonts it does allow. For example, you can put the trademark character in the text, but the Help Compiler deletes it from the help database. Bitmap Magician solves that problem.

When you run it, Bitmap Magician asks you to select a font. When you do, it says that there was an "overflow" with no explanation of what that means. Next, you learn that you are looking at only a demo version. The dialog advises you how to order the real thing. When you acknowledge that piece of good news, the program changes the mouse cursor to an hourglass and leaves it that way. Most Windows users would think that the system is hung up. Not really. You can use the hourglass cursor as if it were an arrow. Close the program, delete its icons from the Program Manager group and proceed to the Help Magician itself.

*Because RoboHelp is a Word template, you can see its source code by opening the macros*

The second annoyance is the overall appearance of Help Magician's windows. The design is an example of a designer gone wild with enthusiasm over 3-D sculptured controls but without the design skills to know how to use them. Everything in the application window and all of the menus and dialog boxes are broadly sculptured. I understand that this is a matter of taste, but I have never seen anything quite like this. There is no menu bar, only a big, fat, sculptured tool bar. When you punch it, it pulls down a menu, also sculptured. The menus use tool buttons. There are the usual File, Edit, Options, Help, and other menus, but they are all represented by ugly tool buttons. The real tool bar is sculpted at the bottom of the window. The overall appearance detracts from the program's functionality.

Help Magician works with its own database format while you compose the help information. Then it converts to RTF format to run the Help Compiler. It launches the word processor of your choice but does not provide templates.

Help Magician has its own editor. That's a good idea, but it's not particularly well implemented. Text that you select for titles and links is surrounded by vertical bars. The vertical bars come in pairs and have to be balanced. You cannot distinguish a starting vertical bar from its terminating vertical bar in a pair. You cannot distinguish two sets of different pairs of vertical bars. A help topic with centered or justified text, a title, and some jump links displays with a mélange of pairs of vertical bars. It's hard to read.

I went through the tutorial process and then tried to add a MIDI sound bite to the tutorial's help document. Somehow I messed up the database. Somehow the MIDI insertion upset the balance of the vertical bars in the line of text. I could not build the RTF file or delete the line. Help Magician stubbornly issued error messages no matter what I tried. Finally, I deleted the entire topic, resulting in lost work.

Next I moved to my own project and imported the RTF file that I built using the manual procedure and Help Author. Without making any changes, I tried to rebuild the Help Magician database into a new RTF file. Help Magician reported another unbalanced marker, this time telling me that I could delete it with a Ctrl+bracket key combination. It didn't tell me that before. I don't know where the unbalanced marker came from. I looked at the original RTF file, and everything looked okay. I deleted the unbalanced marker and saved the RTF file.

Help Magician uses a single-font edit box, and your view of the help text is completely unlike what it is going to look like in a help window, far more so than if you are using Word. Centered text is not centered, and margins are not shown. Those distracting vertical bars are everywhere. To move from topic to topic, you have to change the page number in an edit box at the bottom of the screen and press the Enter key.

There is no preview mode. You can test the database, which displays the help in the same single-font, vertical-bar format and lets you exercise the jumps and popups. However, to see the real thing, you must compile the entire database and run WINHELP.

In one place, the RTF import mangled a graphic token. You could see where some of the RTF protocols were exposed as if they were text. The result was that the Help Compiler could not find the bitmap file. I was able to fix the token in the editor, but the graphic lost its text-centered attribute. I found no way in Help Magician to center or otherwise justify text or tokens. Similarly, there seems to be no way to set margins other than to launch Word, do it from there, and import the RTF file again. Not a good idea, given the import mangling. There were several other places where the import mangled the RTF file. I had to fix them in the Help Magician editor,

and, once again, had no way to set the margins or control the justification.

I launched Word from Help Magician to look at the saved RTF file. It was different now. All of the link phrases and their context identifiers were displayed with a strike-through font and Help Magician had added a bunch of its own footnotes. Even though I had a copy of Word running, Help Magician launched a new copy. (Help Author's launch was smart enough to use the copy of Word that was already running.) The strike-throughs and new footnotes didn't seem to have hurt anything.

Help Magician launches the Help Compiler, WINHELP, Word, HotShot Editor, Paintbrush, and the Sound Recorder. It installs Microsoft Video playback software and shows you how to add audio, video, animation, and MIDI to a help database. Before you use Help Magician on a real project, however, spend some time with its tutorial and get a feel for how it works and where the bugs are. You might like it, and you might not.

### RoboHelp

RoboHelp from Blue Sky runs on top of the word processor. The current release supports Word for Windows only, although Blue Sky plans support for other packages. It won't be an easy port because the main part of RoboHelp is implemented as a Word document template with macros written in the WordBasic programming language. There are some other executable utilities, including one that launches a RoboHelp project, but you can just as easily launch it yourself from Word simply by opening a document that includes the RoboHelp template. This implementation is a dramatic example of what a programmer can do with WordBasic.

The RoboHelp template modifies Word's menus and tool bar and adds a floating tool bar that stays on top and to the right of the document while you are editing. The commands open dialog boxes to establish and change the characteristics of the help project. You add topics, jumps, popups, graphics, search keywords, topic titles, and context mnemonics by pressing tool buttons and filling in the dialogs. RoboHelp manages the document and the help project file and does not use its own database format; it uses the Word document format. One tool button writes the RTF file from the Word document. Another runs the Help Compiler to build the help database.

You can preview a topic by pressing a button, but the preview is not much better than what Word is already showing you. For example, it doesn't show graphics. In fact, RoboHelp's topic preview is

not as good as viewing the topic in Word. The preview justifies all of the text in the left margin regardless of how you have the margins and paragraphs set up. It shows the graphics-insert tokens just as they appear in the document. It's a feature they could have left out.

There are other things that I would change. When you open a RoboHelp document, its Word template defaults the Edit/Find command to search for hidden text, presumably so you can find the jump and pop-up links. Most of my searches are for text in the document, which is not hidden, so I have to change the Find property every time. The "H" tool button that the template added changes selected text to unhidden, an odd choice for this button. I wouldn't think you'd need this one very often. A better choice would be to toggle hidden text into and out of view. Some of the time you want to see the links; other times you want the text to line up more like it does when WINHELP displays it.

Like Help Magician and unlike Help Author, RoboHelp does not use an existing running copy of Word; it launches its own. Because RoboHelp is a Word template, you can see its source code by opening the macros. Furthermore, if you don't like the behavior that I just described or anything else, you can modify it by changing the WordBasic code. You could even add your own adaptation of Help Author's indispensable topic-preview feature.

RoboHelp builds context-identifier files you can include in C++ and Visual Basic programs. It includes a VBX control that adds a help button to a dialog box and prompts you for the associated context identifier. A Screen Capture utility manipulates pictures from the Clipboard. It runs in the background waiting for you to put some graphics in the clipboard. When you do, it pops up with an image-processing tool that lets you modify what you captured into a bitmap for your help database. There is also an icon-composition tool included with RoboHelp.

### Summary

It's not hard to pick a favorite from these alternatives. The manual approach worked, but was tedious. I prefer it over Help Magician, however, which seems to be not quite ready for prime time. Help Author is an order of magnitude better than the manual setup, and RoboHelp is far and away the best tool for the job that I've seen.

When I started this project, I went looking for "Visual Help." Although I didn't find exactly that, I am satisfied that there are tools that make the job easier. I do think, however, that a need for such a product exists. It would have all of the best features of the three packages discussed here. In addition, its editor would emulate the WINHELP display—that's the "visual" part. The tool would use RTF as its native database format and could emulate the jumps, popups, and multimedia features of WINHELP. Unlike Word, it would display graphics without taking all day. Such a program would eliminate Help Compiler until the end of the help document development project. This would be a valuable product. If you build it, they will come.

**DDJ**

To vote for your favorite article, circle inquiry no. 9.

**For More Information**

Microsoft Developer Network CD
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
800-759-5474

The Windows Help Magician
Software Interphase Inc.
82 Cucumber Hill Rd., #113
Foster, RI 02825
401-397-2340

RoboHelp
Blue Sky Software
7486 La Jolla Blvd., Suite 3
La Jolla, CA 92037-9582
800-677-4946

# Algorithms for Directed Graphs

## *A unique approach using genetic algorithms*

Salvatore R. Mangano

Directed graphs underlie any tool that displays a tree diagram, class-relationship diagram, or entity-relationship diagram. As such, you might expect a CASE tool to provide an optimized directed-graph drawer. However, most CASE tools I'm familiar with punt when addressing this problem. Although an algorithm for drawing a directed graph like that shown in Figure 1 is straightforward, a general-purpose graph drawer that draws graphs in an aesthetically pleasing format is difficult to create and computationally expensive. So, CASE tools usually use a few simple rules to get an initial layout and then allow the user to clean things up by dragging objects around. Putting the burden of "pretty drawing" on the user wastes time better spent on the design.

This article looks at a novel solution to this problem using the emerging technology of genetic algorithms (GAs). Specifically, I'll use EOS, my company's C++ GA application framework, and Microsoft's Visual C++ to develop a module for optimizing the aesthetic layout of directed graphs. I'll create a Windows-hosted test application that exercises this module on randomly created graphs. The intent of this article is not to produce a commercial-grade graph drawer, but rather to demonstrate the use of GA technology on a nontrivial and unique problem. Since most programmers' first exposure to GAs is usually on a function-optimization problem, this article provides some insights on the advanced use of GA techniques.

### The Technique

A GA is an algorithm that works with a population of potential solutions to a problem. Through means analogous to biological natural selection, it evolves better and better solutions to the problem. To accomplish this, the user of a GA must first find a way to encode the problem into a string of bits. The bits are analogous to genes and the strings to chromosomes. The encoding of a solution as bit strings is often called the "genotype" and the decoded solution, the "phenotype." The genotype is mapped to the phenotype by the decoding function. The next step after the encoding is the measure of fitness. Fitness is one of the core elements that appear in every variation of a GA. Calculation of fitness involves mapping a solution onto a positive number such that greater numbers correspond to better solutions. This mapping is accomplished by the fitness or objective function.

The second core feature of every genetic algorithm is a population of individuals. At any time during the execution of a genetic algorithm, there exists a population of candidate solutions to the problem (individuals consisting of a genotype and a phenotype). The initial population is usually generated randomly. The process of transforming this initial, mediocre population into a population containing near-optimal solutions is the heart of the GA. It proceeds by iterations of the following genetic operators: selection, reproduction, crossover, and mutation.

Selection is the process by which candidate individuals from the current generation are chosen to produce the next generation. Selection is a survival-of-the-fittest strategy. After two individuals are selected, a weighted coin is flipped to determine if the individuals will mate to produce two new offspring or simply be placed in the next generation as is. Mating is accomplished by the crossover operation. The probability of mating is called the "crossover probability" (pc). The simplest form of crossover, called "single point," is shown in Figure 2. As each bit is copied from parent to child, it is subject to mutation based on the mutation probability (pm). Pm is usually very small, relative to pc. Iterations of selection, reproduction, and mating are repeated until a new population is created. At this point, the fitness values of the new population are recalculated, and the process repeats until either some acceptable solution is found or an upper time limit has been reached. The GA described above can be summarized by the procedure shown in Figure 3.

### The Tools

A GA framework is useful due to the large number of GA variants that can be produced by altering one or many of the steps in the basic algorithm. GA re-

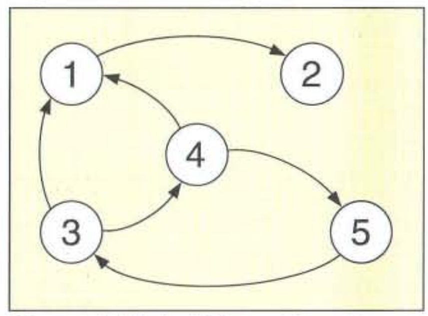



***Figure 1:*** *Typical directed graph.*

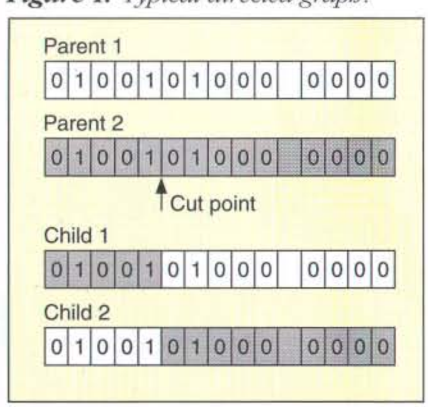



***Figure 2:*** *Single-point crossover.*

*Sal is president of Man Machine Interfaces. He can be reached at 555 Broad Hollow Road, Suite 230, Melville, NY 11747 or on CompuServe at 72053,2032.*

```
Initialize a random population and measure its fitness.
WHILE (stopping criteria has not been reached)
BEGIN
   WHILE (next generation is not full)
   BEGIN
      Select 2 parents randomly based on fitness.
      IF (Flip(pc)) THEN
         Cross parents (mutating with probability pm)
         and place children in next generation.
      ELSE
         Place parents into next generation untouched.
   END
END

Solution with highest fitness is the answer.
```

***Figure 3:*** *Pseudocode detailing the GA process.*

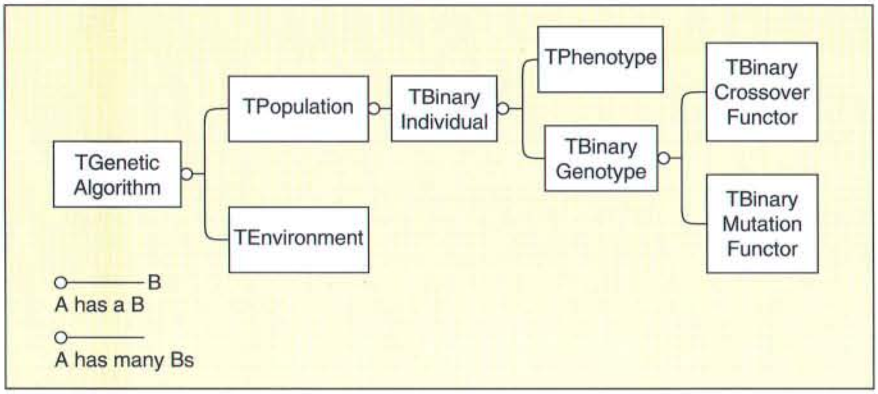



***Figure 4:*** *Relationship between classes.*

*(continued from page 92)*
searchers have invented several variations on selection, reproduction, crossover, and mutation. Each variation can be mixed and matched to produce a unique GA variant. Object orientation turns out to be an ideal technique for expressing these variations. Through an adept combination of inheritance and composition, all the GA variants can be expressed. This ultimately allows you to code a GA using the basic technique and then try variations by instantiating different classes.

Although EOS consists of over 80 classes, I'll restrict this discussion to a small relevant subset—*TBasicGA*, *TPopulation*, *TEnvironment*, *TBinaryIndividual*, *TGenotype*, *TPhenotype*, *TBinaryCrossoverFunctor*, and *TBinaryMutationFunctor*. These bases consist of many derived classes that implement variants of the basic GA. Other classes exist to implement special-purpose features. Each of the classes listed encapsulates a different behavior of the overall GA. *TBasicGA* is the genetic-algorithm interface class. *TPopulation* is a collection class that holds instances of *TIndividual*. *TPopulation* encapsulates the selection operation. *TEnvironment* encapsulates the GA's parameters—pc, pm, the random-number generator, and other statistical information. *TBinaryIndi-*

*vidual* is an interface class that unifies instances of *TGenotype* and *TPhenotype* into a single object. *TBinaryGenotype* encapsulates the binary genetic coding of the problem as strings, and it provides an interface to the crossover and mutation classes. *TPhenotype* encapsulates the decoding function and the fitness function. It is the main class from which you derive to build a GA-based application. *TBinaryCrossoverFunctor* and *TBinaryMutationFunctor* are classes that encapsulate the operations of crossover and mutation. These classes are called functors because they are functional objects. Functors are used so various flavors of crossover and mutation can be plugged in or out of a genotype without recoding any of the genotype's methods. Figure 4 shows the relationship between these classes.

### The GA Module

To derive a genetic encoding, I'll formalize the problem we are attempting to solve. We are given some arbitrary directed graph, as well as a grid where each cell represents a potential home for the graphical depiction of a node in the graph. The goal is to find an assignment of nodes to cells such that when the arcs are drawn, we get an aesthetically pleasing picture. Stating the problem in this way makes some crucial assumptions that may not be true in a real situation. First, I assume that the nodes, when drawn, are of equal size. Second, I assume that once I have assigned nodes to cells, the arcs can easily be drawn to complete the best possible drawing. (In other words, we need not optimize the drawing of arcs.) Third, I assume that the nodes are equally spaced in a grid and not arbitrarily placed on the output screen. I make these assumptions to simplify the example and the code. A more general solution is certainly possible using GAs.

### Genetic Encoding

If each node in the graph is assigned a sequential number, then the problem can be viewed as a mapping of each node number onto an (x, y) coordinate in the grid. The mapping that keeps connected nodes close together and produces the fewest arc crossings will yield more aesthetically pleasing drawings. Other domain-dependent criteria may come into play when determining better drawings, but I ignore this possibility to expedite the solution.

Given the above formalism, the encoding treats the bit string as a series of (x, y) pairs. The first pair assigns node0 to grid cell (x0, y0). The second assigns node1 to (x1, y1), and so on. This encoding allows collisions (two or more nodes are assigned to a single cell), so I need a collision-resolution procedure. A problem like this often arises in GAs, and there are several approaches to handling it. Some programmers assign very low fitness values to illegal genotypes. Others attempt to repair illegal genotypes before decoding them. Still others create special-purpose crossover and mutation operators that do not allow illegal genotypes to arise in the first place. In this case, I'll resolve a collision by searching for the closest empty cell to the one assigned, according to a fixed procedure. This is similar to a repair technique, but we are repairing the phenotype instead of the genotype.

Given that I have a graph with N nodes and a grid that is X cells wide and Y cells high, I can calculate the required length of the bit string using the equation in Figure 5. If X and Y are not powers of 2, then it is possible that (x, y) pairs can be encoded such that either x or y is greater then X or Y. I resolve this problem by always decoding x modulo X and y modulo Y. The decoding is implemented by a *TPhenotype* derived class called *CGraphDrawingPheno*. This class

$$\text{Chromosome Length} = N * \left( \lceil \log_2(X) \rceil + \lceil \log_2(Y) \rceil \right)$$

***Figure 5:*** *Equation to calculate the required length of a bit string.*

contains a two-dimensional matrix that will represent the grid. The genotype will be decoded so that each entry in the matrix will receive the node number of the node assigned to that grid position. Empty positions will be assigned

> *A GA framework is useful due to the large number of GA variants that can be produced*

a value of 0. The decoding of the genotype is implemented by *CGraphDrawingPheno::Decode()* member function; see Listings One and Two, page 106. This function uses a reference to a graph drive class to determine the number of bits in each component in the encoding. It copies these bits to buffers to be converted to integers by the utility functions *AllelesToInt()*. "Allele," a term borrowed from biology, refers to the value expressed by a gene. Once the node, its row, and column in the grid are decoded, the member function *GetNearestEmptyCell()* is called to resolve the possibility of a node already existing at the desired location.

### The Fitness Function

Now that I have a way to encode the placement of a graph's nodes on a grid, I need a technique for evaluating each placement's fitness. There are many ways to do this, depending on what you consider to be an aesthetically pleasing layout. When deriving this fitness function, I let intuition guide me in the initial derivation and then experiment to tweak the function so it works well for a variety of graphs. The function I ultimately arrived at can be seen in the *CGraphDrawingPheno* class's *CalcFitness()* member function; see Listing Two. The idea behind this function is to reward genotypes that decode into drawings where nodes connected by an arc are adjacent or close and to penalize when nodes are adjacent but not connected. This is done on a node-by-node basis so the resulting fitness value is a measure of how well nodes of the entire graph were assigned to grid locations. Notice that I completely ignore arc drawing for simplicity. The remaining members of *CGraphDrawingPheno* implement construction, destruction and copying. I also include some private-utility functions that encapsulate the testing

for adjacency and the calculation of distance between grid cells. These can be used in experimenting with variations of the fitness function.

I include three other classes in this module: *CGAGraphDriver, CGraphDrawerGA*, and *CWordMatrix. CGAGraphDriver* is an interface class that collects information (such as number of nodes in the graph and the size of the grid) before the GA and its associated objects are initialized; see Listings Three and Four (page 107). A very important function in this class is *CalcChromosomeLength( )*, which, based on the number of nodes and the size of the grid, determines the number of bits necessary in a chromosome to encode the problem. Also included in this class are functions for drawing the optimized and unoptimized views of the graph. These use Windows-specific functions, but the logic can be easily ported to other graphics systems.

*CGraphDrawerGA* is derived from the EOS class *TBasicGA*. It overrides the population-creation function and several reporting functions useful for testing the GA's performance before it is embedded into a larger application. Listings Five and Six (page 147) show the class declaration and implementation of *CGraphDrawerGA*. The important function here is *CreatePopulation()*. This function determines the characteristics of the genotype, phenotype, and the population. I use a two-point crossover-operator genotype (instead of single point) because this tends to work better with longer chromosomes. I also use a population technique known as *Elitism*, which ensures that a certain number (in our case, two) of the best individuals from the previous generation make it to the next generation. This improves performance on some types of problems.

The utility class *CWordMatrix* implements a 2-D matrix of WORDS. Its implementation makes use of the MFC's *CObArray* and *CWordArray* to create an array of *CWordArrays*.

### The Test Program

To test the graph-optimizing GA, I created a simple Windows-hosted application using Visual C++. I used AppStudio, AppWizard, and ClassWizard to automate the creation of this program. The program uses two dialog boxes. The first dialog box allows me to specify the graph logically in terms of the number of nodes and their connections. The second allows me to specify the bounds of the grid and trigger the GA optimization of the graph on the grid. I also include options for displaying the optimized and unoptimized views of the graph. The optimized view is the best solution found by the GA; the unoptimized view is an arbitrary drawing of the graph from first node to the last. Due to the length of the test program, the complete listings for this project are available electronically; see "Availability," page 3.

### Conclusion

Experiments that I have conducted using the GA-based graph-drawing module demonstrate that GAs present a viable solution to the problem. By improving the fitness function and by possible use of custom genetic operators, I believe that this technique will also work in commercial CASE tools.

### References


Coplien, James O. *Advanced C++: Programming Styles and Idioms.* Reading, MA: Addison-Wesley, 1992.

Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs.* New York, NY: Springer-Verlag, 1992.


**DDJ**


**(Listings begin on page 106.)**


To vote for your favorite article, circle inquiry no. 10.

## Listing One *(Text begins on page 38.)*

```
/* Blowfish.h */

#define MAXKEYBYTES 56              /* 448 bits */

short opensubkeyfile(void);
unsigned long F(unsigned long x);
void Blowfish_encipher(unsigned long *xl, unsigned long *xr);
void Blowfish_decipher(unsigned long *xl, unsigned long *xr);
short InitializeBlowfish(char key[], short keybytes);
```

## Listing Two

```
/* Blowfish.c */

#include <dos.h>
#include <graphics.h>
#include <io.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <alloc.h>
#include <ctype.h>
#include <dir.h>
#include <bios.h>
#include <Types.h>

#define  N  16
#define  noErr 0
#define  DATAERROR -1
#define  KEYBYTES 8
#define  subkeyfilename "Blowfish.dat"

static unsigned long P[18];
static unsigned long S[4,256];
static FILE*          SubkeyFile;

short opensubkeyfile(void) /* read only */
{
   short error;
   error = noErr;
   if((SubkeyFile = fopen(subkeyfilename,"rb")) == NULL) {
      error = DATAERROR;
   }
   return error;
}
unsigned long F(unsigned long x)
{
   unsigned short a;
   unsigned short b;
   unsigned short c;
   unsigned short d;
   unsigned long  y;

   d = x & 0x00FF;
   x >>= 8;
   c = x & 0x00FF;
   x >>= 8;
   b = x & 0x00FF;
   x >>= 8;
   a = x & 0x00FF;

   y = ((S[0, a] + (S[1, b] % 32)) ^ S[2, c]) + (S[3, d] % 32);
   /* Note: There is a good chance that the following line will execute faster */
   /* y = ((S[0,a] + (S[1, b] & 0x001F)) ^ S[2, c]) + (S[3,d] & 0x001F); */
   return y;
}
void Blowfish_encipher(unsigned long *xl, unsigned long *xr)
{
   unsigned long  Xl;
   unsigned long  Xr;
   unsigned long  temp;
   short          i;

   Xl = *xl;
   Xr = *xr;
   for (i = 0; i < N; ++i) {
      Xl = Xl ^ P[i];
      Xr = F(Xl) ^ Xr;

      temp = Xl;
      Xl = Xr;
      Xr = temp;
   }
   temp = Xl;
   Xl = Xr;
   Xr = temp;

   Xr = Xr ^ P[N];
   Xl = Xl ^ P[N + 1];

   *xl = Xl;
   *xr = Xr;
}
void Blowfish_decipher(unsigned long *xl, unsigned long *xr)
{
   unsigned long  Xl;
   unsigned long  Xr;
   unsigned long  temp;
   short          i;

   Xl = *xl;
   Xr = *xr;
```

```
    for (i = N + 1; i > 1; --i) {
       Xl = Xl ^ P[i];
       Xr = F(Xl) ^ Xr;

       /* Exchange Xl and Xr */
       temp = Xl;
       Xl = Xr;
       Xr = temp;
    }
    /* Exchange Xl and Xr */
    temp = Xl;
    Xl = Xr;
    Xr = temp;

    Xr = Xr ^ P[1];
    Xl = Xl ^ P[0];

    *xl = Xl;
    *xr = Xr;
}
short InitializeBlowfish(char key[], short keybytes)
{
    short          i;
    short          j;
    short          k;
    short          error;
    short          numread;
    unsigned long  data;
    unsigned long  datal;
    unsigned long  datar;
    /* First, open the file containing the array initialization data */
    error = opensubkeyfile();
    if (error == noErr) {
       for (i = 0; i < N + 1; ++i) {
          numread = fread(&data, 4, 1, SubkeyFile);
          printf("%d : %d : %.4s\n", numread, i, &data);
          if (numread != 1) {
             return DATAERROR;
          } else {
             P[i] = data;
          }
       }
       for (i = 0; i < 4; ++i) {
          for (j = 0; j < 256; ++j) {
             numread = fread(&data, 4, 1, SubkeyFile);
             printf("[%d, %d] : %.4s\n", i, j, &data);
             if (numread != 1) {
                return DATAERROR;
             } else {
                S[i, j] = data;
             }
          }
       }
       fclose(SubkeyFile);
       j = 0;
       for (i = 0; i < 18; ++i) {
          data = 0x00000000;
          for (k = 0; k < 4; ++k) {
             data = (data << 8) | key[j];
             j = j + 1;
             if (j > keybytes) {
                j = 0;
             }
          }
          P[i] = P[i] ^ data;
       }
       datal = 0x00000000;
       datar = 0x00000000;
       for (i = 0; i < 18; i += 2) {
          Blowfish_encipher(&datal, &datar);

          P[i] = datal;
          P[i + 1] = datar;
       }
       for (j = 0; i < 4; ++j) {
          for (i = 0; i < 256; i += 2) {
             Blowfish_encipher(&datal, &datar);

             S[j, i] = datal;
             S[j, i + 1] = datar;
          }
       }
    } else {
       printf("Unable to open subkey initialization file : %d\n", error);
    }
    return error;
}
```

**End Listings**

## Listing One *(Text begins on page 44.)*

```
/* CONVOLVE.C */
/* Copyright (C) 1993 Mac A. Cody - All rights reserved */

#include "convolve.h"

/* DualConvDec2 - Convolution of data array with decomposition filters
                followed by decimation by two.
   Input(s): REAL_TYPE *src     - Pointer to source data sample array.
             REAL_TYPE *htilda  - Pointer to lowpass filter array.
             REAL_TYPE *gtilda  - Pointer to highpass filter array.
             short srclen       - length of source data array.
             short filtlen      - length of filter arrays.
   Output(s): REAL_TYPE *adst   - Pointer to approximation data sample array.
              REAL_TYPE *ddst   - Pointer to detail data sample array.
*/
void DualConvDec2(REAL_TYPE *src, REAL_TYPE *adst, REAL_TYPE *ddst,
          REAL_TYPE *htilda, REAL_TYPE *gtilda, short srclen, short filtlen)
{
  short i, j, brklen, endlen;
  REAL_TYPE adot_p, ddot_p;
  REAL_TYPE *head_src, *lp_fltr, *hp_fltr;
  brklen = 1; /* initial break in dot product is after first element */
  /* perform truncated dot products until filter no longer hangs off end of
     array; decimation by two handled by two-element shift; break count
     increases by two on every iteration */
  for(j = 0; j < filtlen; j += 2, brklen += 2)
  {
    head_src = src + j; /* point to appropriate initial element at head */
    lp_fltr = htilda;   /* set up pointer to lowpass filter */
    hp_fltr = gtilda;   /* set up pointer to highpass filter */
    adot_p = *head_src * *lp_fltr++;   /* initial lowpass product of head */
    ddot_p = *head_src-- * *hp_fltr++; /* initial highpass product of head */
    for(i = 1; i < brklen; i++) /* perform remaining products of head */
    {
      adot_p += *head_src * *lp_fltr++;
      ddot_p += *head_src-- * *hp_fltr++;
    }
    *adst++ = adot_p; /* save the completed lowpass dot product */
    *ddst++ = ddot_p; /* save the completed highpass dot product */
  }
  endlen = srclen + filtlen - 2; /* find total length of array */
  /* perform convolution to the physical end of the array
     with a simple dot product loop */
  for(; j <= endlen; j += 2)
  {
    head_src = src + j; /* point to appropriate initial element at head */
    lp_fltr = htilda;   /* set up pointer to lowpass filter */
    hp_fltr = gtilda;   /* set up pointer to highpass filter */
    adot_p = *head_src * *lp_fltr++;   /* initial lowpass product */
    ddot_p = *head_src-- * *hp_fltr++; /* initial highpass product */
    for(i = 1; i < filtlen; i++) /* perform remaining products */
    {
      adot_p += *head_src * *lp_fltr++;
      ddot_p += *head_src-- * *hp_fltr++;
    }
    *adst++ = adot_p; /* save the completed lowpass dot product */
    *ddst++ = ddot_p; /* save the completed highpass dot product */
  }
  /* perform convolution off the physical end of the array
     with a partial dot product loop, like at the beginning */
  for(brklen = filtlen - 2, j = 2; brklen > 0; brklen -= 2, j += 2)
  {
    head_src = src + endlen; /* point to last element in array */
    lp_fltr = htilda + j;    /* set up pointer to lowpass filter offset */
    hp_fltr = gtilda + j;    /* set up pointer to highpass filter offset*/
    adot_p = *head_src * *lp_fltr++;   /* initial lowpass product */
    ddot_p = *head_src-- * *hp_fltr++; /* initial highpass product */
    for(i = 1; i < brklen; i++) /* perform remaining products */
    {
      adot_p += *head_src * *lp_fltr++;
      ddot_p += *head_src-- * *hp_fltr++;
    }
    *adst++ = adot_p; /* save the completed lowpass dot product */
    *ddst++ = ddot_p; /* save the completed highpass dot product */
  }
} /* End of DualConvDec2 */
/* DualConvInt2Sum - Convolution of data array with reconstruction
                   filters with interpolation by two and sum together.
   Input(s): REAL_TYPE *asrc    - Pointer to approximation data sample array.
             REAL_TYPE *dsrc    - Pointer to detail data sample array.
             REAL_TYPE *h       - Pointer to lowpass filter array.
             REAL_TYPE *g       - Pointer to highpass filter array.
             short srclen       - length of source data array.
             short filtlen      - length of filter arrays.
   Output(s): REAL_TYPE *dst   - Pointer to output data sample array.
*/
void DualConvInt2Sum(REAL_TYPE *asrc, REAL_TYPE *dsrc, REAL_TYPE *dst,
                     REAL_TYPE *h, REAL_TYPE *g, short srclen, short filtlen)
{
  short i, j, endlen;
  REAL_TYPE dot_pe, dot_po;
  REAL_TYPE *head_asrc, *head_dsrc, *lp_fltr, *hp_fltr;
  endlen = srclen + filtlen - 2; /* find total length of array */
  filtlen /= 2;          /* adjust filter length value for interpolation */
  j = filtlen - 1;       /* start with filter 'covering' end of array */
  head_asrc = asrc + j; /* point to initial element at head */
  head_dsrc = dsrc + j; /* point to initial element at head */
  lp_fltr = h + 1;       /* set up pointer to lowpass filter */
  hp_fltr = g + 1;       /* set up pointer to highpass filter */
  /* initial lowpass and highpass odd product */
  dot_po = *head_asrc-- * *lp_fltr + *head_dsrc-- * *hp_fltr;
  lp_fltr += 2;  hp_fltr += 2; /* skip over even filter elements */
  for(i = 1; i < filtlen; i += 2) /* perform remaining products */
  {
    dot_po += *head_asrc-- * *lp_fltr + *head_dsrc-- * *hp_fltr;
    lp_fltr += 2;  hp_fltr += 2; /* skip over even filter elements */
  }
  /* save the completed lowpass and highpass odd dot product */
  *dst++ = dot_po;
  /* perform initial convolution with a simple dot product loop */
  for(j++; j <= endlen; j++)
  {
    head_asrc = asrc + j; /* point to appropriate initial element at head */
    head_dsrc = dsrc + j; /* point to appropriate initial element at head */
    lp_fltr = h;          /* set up pointer to lowpass filter */
    hp_fltr = g;          /* set up pointer to highpass filter */
    /* initial lowpass and highpass even product */
    dot_pe = *head_asrc * *lp_fltr++ + *head_dsrc * *hp_fltr++;
    /* initial lowpass and highpass odd product */
    dot_po = *head_asrc-- * *lp_fltr++ + *head_dsrc-- * *hp_fltr++;
    for(i = 1; i < filtlen; i++) /* perform remaining products */
    {
      dot_pe += *head_asrc * *lp_fltr++ + *head_dsrc * *hp_fltr++;
      dot_po += *head_asrc-- * *lp_fltr++ + *head_dsrc-- * *hp_fltr++;
    }
    /* save the completed lowpass and highpass even dot product */
    *dst++ = dot_pe;
    /* save the completed lowpass and highpass odd dot product */
    *dst++ = dot_po;
  }
} /* End of DualConvInt2Sum */
```

## Listing Two

```
/* AWPT.C */
/* Copyright (C) 1993 Mac A. Cody - All rights reserved */

#include "wp_types.h"
#include "awpt.h"
#include "convolve.h"

/* AWPT - Aperiodic Wavelet Packet Transform: Data is assumed to be
          non-periodic, so convolutions do not wrap around arrays.
          Convolution data past end of data is generated and retained
          to allow perfect reconstruction of original input.
   Input(s): REAL_TYPE *indata - Pointer to input data sample array.
             REAL_TYPE *htilda - Pointer to lowpass filter array.
             REAL_TYPE *gtilda - Pointer to highpass filter array.
             short filtlen     - Length of filter arrays.
   Output(s): WPTstruct *out - Pointer to transform data structure.
   Note: Structure pointed to by 'out' contains:
         out->levels - Number of levels in transform (short).
         out->length - Length of input data sample array (short).
         out->data   - Pointer to pointer of arrays of data (REAL_TYPE ***).
*/
void AWPT(REAL_TYPE *indata, WPTstruct *out,
          REAL_TYPE *htilda, REAL_TYPE *gtilda, short filtlen)
{
  short i, j, nodes, datalen;
  REAL_TYPE *src;
  levels = out->levels;
  datalen = out->length; /* start with length of input array */
  /* loop for all levels, halving the data length on each lower level */
  for (i = 0, nodes = 2; i < levels; i++, nodes <<= 1)
  {
    for(j = 0; j < nodes; j += 2)
    {
      if(out->data[i][j] == 0) continue;
      if(i == 0)        /* ... source for highest level is input data */
        src = indata;
      else              /* ... source is corresponding node of higher level */
        src = out->data[i-1][j >> 1];
      DualConvDec2(src, out->data[i][j], out->data[i][j+1],
                                htilda, gtilda, datalen, filtlen);
    }
    datalen /= 2; /* input length for next level is half this level */
  }
} /* End of AWPT */
/* IAWPT - Inverse Aperiodic Wavelet Packet Transform: Data is assumed to be
           non-periodic, so convolutions do not wrap arround arrays.
           Convolution data past end of data is used to generate perfect
           reconstruction of original input.
   Input(s): WPTstruct *in     - Pointer to transform data structure.
             REAL_TYPE *htilda - Pointer to lowpass filter array.
             REAL_TYPE *gtilda - Pointer to highpass filter array.
             short filtlen     - Length of filter arrays.
   Note: Structure pointed to by 'in' contains:
         in->levels - Number of levels in transform (short).
         in->length - Length of output data sample array (short).
         in->data   - Pointer to pointer of arrays of data (REAL_TYPE ***).
   Output(s): REAL_TYPE *indata - Pointer to input data sample array.
*/
void IAWPT(WPTstuct *in, REAL_TYPE *outdata,
           REAL_TYPE *htilda, REAL_TYPE *gtilda, short filtlen)
{
  short i, j, levels, nodes, datalen;
  REAL_TYPE *dst;
  levels = in->levels;
  /* start with length of lowest level input array */
  datalen = in->length >> levels;
  /* loop for all levels, doubling the data length on each higher level;
     destination of all but the highest branch of the reconstruction
     is the next higher node */
  for (i = levels - 1, nodes = 1 << levels; i >= 0; i--, nodes >>= 1)
  {
    for(j = 0; j < nodes; j += 2)
    {
      if(out->data[i][j] == 0) continue;
      if(i == 0) /* ... destination for highest level is input data */
        dst = outdata;
      else       /* ... destination is corresponding node of higher level */
        dst = in->data[i - 1][j >> 1];
      DualConvInt2Sum(in->data[i][j], in->data[i][j+1], dst,
                                 htilda, gtilda, datalen, filtlen);
    }
    datalen *= 2; /* input length for next level is half this level */
  }
}/* End of IAWPT */
```

**End Listings**

## Listing One *(Text begins on page 56.)*

```c
/* Update to Greg Viot's fuzzy system -- DDJ, February 1993, page 94 */
/* By J. Tucker, P. Fraley, and L. Swanson, April 1993 */
#include <stdio.h>
#include <string.h>                         /* NEW */
# define max(a,b)   (a<b ? b : a)           /* NEW */
# define min(a,b)   (a>b ? b : a)           /* NEW */
struct io_type *System_Inputs;              /* anchor inputs NEW */
struct io_type *System_Output;              /* anchor output NEW */
#define MAXNAME 10
#define UPPER_LIMIT 255
struct io_type{
   char name[MAXNAME];
   int value;
   struct mf_type *membership_functions;
   struct io_type *next;
   };
struct mf_type{
   char name[MAXNAME];
   int value;
   int point1;
   int point2;
   float slope1;
   float slope2;
   struct mf_type *next;
   };
struct rule_type{
   struct rule_element_type *if_side;
   struct rule_element_type *then_side;
   struct rule_type *next;
   };
struct rule_element_type{
   int *value;
   struct rule_element_type *next;
   };
struct rule_type *Rule_Base;

main(argc,argv)                             /* NEW */
int argc;                                   /* NEW */
char *argv[];                               /* NEW */
{  int input1, input2;                      /* NEW */
   if(argc!=3)                              /* NEW */
   {  printf("Error - Must supply 2 numeric inputs.\n"); /* NEW */
      printf("        Inputs scaled to range 0-255.\n"); /* NEW */
      printf("Usage: %s angle velocity\n",argv[0]);      /* NEW */
      exit(0);                              /* NEW */
   }                                        /* NEW */
   input1=atoi(argv[1]);                    /* NEW */
   input2=atoi(argv[2]);                    /* NEW */
   initialize_system();                     /* Read input files, NEW */
   get_system_inputs(input1,input2);        /* Get & put argv NEW */
   fuzzification();
   rule_evaluation();
   defuzzification();
   put_system_outputs();                    /* print all data, NEW */
}                                           /* END MAIN */
fuzzification()
{  struct io_type *si;
   struct mf_type *mf;
   for(si=System_Inputs;si!=NULL;si=si->next)
      for(mf=si->membership_functions;mf!=NULL;mf=mf->next)
     compute_degree_of_membership(mf,si->value);
}                                           /* END FUZZIFICATION */
rule_evaluation()
{  struct rule_type *rule;
   struct rule_element_type *ip;      /* if ptr */
   struct rule_element_type *tp;      /* then ptr */
   int strength;
   int nomatch=0;                           /* NEW, test some rules */
   for(rule=Rule_Base;rule!=NULL;rule=rule->next)
   {  strength=UPPER_LIMIT;
      for(ip=rule->if_side;ip!=NULL;ip=ip->next)
            strength=min(strength,*(ip->value));
      for(tp=rule->then_side;tp!=NULL;tp=tp->next)
      {  *(tp->value)=max(strength,*(tp->value));       /* NEW */
         if(strength>0)nomatch=1;                       /* NEW */
      }                                                 /* NEW */
   }
   if(nomatch==0)printf("NO MATCHING RULES FOUND!\n"); /* NEW */
}                                           /* END RULE EVALUATION */
defuzzification()
{  struct io_type *so;
   struct mf_type *mf;
   int sum_of_products;
   int sum_of_areas;
   int area, centroid;
   for(so=System_Output;so!=NULL;so=so->next)
   {  sum_of_products=0;
      sum_of_areas=0;
      for(mf=so->membership_functions;mf!=NULL;mf=mf->next)
      {  area=compute_area_of_trapezoid(mf);
         centroid=mf->point1+(mf->point2-mf->point1)/2;
         sum_of_products+=area*centroid;
         sum_of_areas+=area;
      }
      if(sum_of_areas==0)                                  /* NEW */
      {  printf("Sum of Areas = 0, will cause div error\n"); /* NEW */
         printf("Sum of Products= %d\n",sum_of_products);  /* NEW */
         so->value=0;                                      /* NEW */
         return;                                           /* NEW */
      }                                                    /* NEW */
      so->value=sum_of_products/sum_of_areas;
   }
}                                           /* END DEFUZZIFICATION */
compute_degree_of_membership(mf,input)
struct mf_type *mf;
int input;
{  int delta_1, delta_2;
   delta_1=input - mf->point1;
   delta_2=mf->point2 - input;
   if((delta_1<=0)||(delta_2<=0))mf->value=0;
   else
   {  mf->value=min((mf->slope1*delta_1),(mf->slope2*delta_2));
      mf->value=min(mf->value,UPPER_LIMIT);
   }
}                                           /* END DEGREE OF MEMBERSHIP */
compute_area_of_trapezoid(mf)
struct mf_type *mf;
{  float run_1,run_2,area,top;
   float base;
   base=mf->point2 - mf->point1;
   run_1=mf->value / mf->slope1;
   run_2=mf->value / mf->slope2;
   top=base - run_1 - run_2;
   area=mf->value*(base+top)/2;
   return(area);
}                                           /* END AREA OF TRAPEZOID */
initialize_system()                         /* NEW FUNCTION INITIALIZE */
{  int a, b, c, d, x;
   char buff[10],buff1[4],buff2[4];
   static char filename1[]="in1";  /* "angles" filename */
   static char filename2[]="in2";  /* "velocities" filename */
   static char filename3[]="out1"; /* "forces" filename */
   FILE *fp;
   struct io_type *outptr;
   struct mf_type *top_mf;
   struct mf_type *mfptr;
   struct io_type *ioptr;
   struct rule_type *ruleptr;
   struct rule_element_type *ifptr;
   struct rule_element_type *thenptr;
   ioptr=NULL;
   ruleptr=NULL;
   ifptr=NULL;
   thenptr=NULL;
/* READ THE FIRST FUZZY SET (ANTECEDENT); INITIALIZE STRUCTURES */
   if((fp=fopen(filename1,"r"))==NULL)   /* open "angles" file */
   {  printf("ERROR- Unable to open data file named %s.\n",filename1);
      exit(0);
   }
   ioptr=(struct io_type *)calloc(1,sizeof(struct io_type));
   System_Inputs=ioptr;                     /* Anchor to top of inputs */
   x=fscanf(fp,"%s",buff);                  /* from 1st line, get set's name */
   sprintf(ioptr->name,"%s",buff);          /* into struct io_type.name */
   mfptr=NULL;
   while((x=fscanf(fp,"%s %d %d %d %d",buff,&a,&b,&c,&d))!=EOF)/* get line */
   {  if(mfptr==NULL)                       /* first time thru only */
      {  mfptr=(struct mf_type *)calloc(1,sizeof(struct mf_type));
         top_mf=mfptr;
         ioptr->membership_functions=mfptr;
      }
      else
      {  for(mfptr=top_mf;mfptr->next;mfptr=mfptr->next); /* spin to last */
```

*(continued on page 102)*

**Listing One** *(Listing continued, text begins on page 56.)*

```
        mfptr->next=(struct mf_type *)calloc(1,sizeof(struct mf_type));
        mfptr=mfptr->next;
     }
     sprintf(mfptr->name,"%s",buff);    /* membership name, NL, ZE, etc */
     mfptr->point1=a;                   /* left x axis value */
     mfptr->point2=d;                   /* right x axis value */
     if(b-a>0) mfptr->slope1=UPPER_LIMIT/(b-a);     /* left slope */
     else
     {  printf("Error in input file %s, membership element %s.\n",
        filename1,buff);
        exit(1);
     }
     if(d-c>0) mfptr->slope2=UPPER_LIMIT/(d-c);     /* right slope */
     else
     {  printf("Error in input file %s, membership element %s.\n",
        filename1,buff);
        exit(1);
     }
  }
  close(fp);                            /* close "angles" file */
/* READ THE SECOND FUZZY SET (ANTECEDENT); INITIALIZE STRUCTURES */
  if((fp=fopen(filename2,"r"))==NULL)   /* open "velocity" file */
  {  printf("ERROR- Unable to open data file named %s.\n",filename2);
     exit(0);
  }
  ioptr->next=(struct io_type *)calloc(1,sizeof(struct io_type));
  ioptr=ioptr->next;
  x=fscanf(fp,"%s",buff);               /* from 1st line, get set's name */
  sprintf(ioptr->name,"%s",buff);       /* into struct io_type.name */
  mfptr=NULL;
  while((x=fscanf(fp,"%s %d %d %d %d",buff,&a,&b,&c,&d))!=EOF)/* get line */
  {  if(mfptr==NULL)                    /* first time thru only */
     {  mfptr=(struct mf_type *)calloc(1,sizeof(struct mf_type));
        top_mf=mfptr;
        ioptr->membership_functions=mfptr;
     }
     else
     {  for(mfptr=top_mf;mfptr->next;mfptr=mfptr->next); /* spin to last */
        mfptr->next=(struct mf_type *)calloc(1,sizeof(struct mf_type));
        mfptr=mfptr->next;
     }
     sprintf(mfptr->name,"%s",buff);    /* membership name, NL, ZE, etc */
     mfptr->point1=a;                   /* left x axis value */
     mfptr->point2=d;                   /* right x axis value */
     if(b-a>0) mfptr->slope1=UPPER_LIMIT/(b-a);     /* left slope */
     else
     {  printf("Error in input file %s, membership element %s.\n",
        filename2,buff);
        exit(1);
     }
     if(d-c>0) mfptr->slope2=UPPER_LIMIT/(d-c);     /* right slope */
     else
     {  printf("Error in input file %s, membership element %s.\n",
        filename2,buff);
        exit(1);
     }
  }
  close(fp);                            /* close "velocity" file */
/* READ THE THIRD FUZZY SET (CONSEQUENCE); INITIALIZE STRUCTURES */
  if((fp=fopen(filename3,"r"))==NULL)   /* open "force" file */
  {  printf("ERROR- Unable to open data file named %s.\n",filename3);
     exit(0);
  }
  ioptr=(struct io_type *)calloc(1,sizeof(struct io_type));
  System_Output=ioptr;                  /* Anchor output structure */
  x=fscanf(fp,"%s",buff);               /* from 1st line, get set's name */
  sprintf(ioptr->name,"%s",buff);       /* into struct io_type.name */
  mfptr=NULL;
  while((x=fscanf(fp,"%s %d %d %d %d",buff,&a,&b,&c,&d))!=EOF)/* get line */
  {  if(mfptr==NULL)                    /* first time thru */
     {  mfptr=(struct mf_type *)calloc(1,sizeof(struct mf_type));
        top_mf=mfptr;
        ioptr->membership_functions=mfptr;
     }
     else
     {  for(mfptr=top_mf;mfptr->next;mfptr=mfptr->next);
        mfptr->next=(struct mf_type *)calloc(1,sizeof(struct mf_type));
        mfptr=mfptr->next;
     }
     sprintf(mfptr->name,"%s",buff);    /* membership name, NL, ZE, etc */
     mfptr->point1=a;                   /* left x axis value */
     mfptr->point2=d;                   /* right x axis value */
     if(b-a>0) mfptr->slope1=UPPER_LIMIT/(b-a);     /* left slope */
     else
     {  printf("Error in input file %s, membership element %s.\n",
        filename3,buff);
        exit(1);
     }
     if(d-c>0) mfptr->slope2=UPPER_LIMIT/(d-c);     /* right slope */
     else
     {  printf("Error in input file %s, membership element %s.\n",
        filename3,buff);
        exit(1);
     }
  }
  close(fp);                            /* close "force" file */
/* READ RULES FILE; INITIALIZE STRUCTURES */
  ioptr=NULL;
  outptr=NULL;
  if((fp=fopen("rules","r"))==NULL)     /* open rules file */
  {  printf("ERROR- Unable to open data file named %s.\n","rules");
     exit(0);
  }
  ruleptr=(struct rule_type *)calloc(1,sizeof(struct rule_type));
  if(ioptr==NULL)Rule_Base=ruleptr;     /* first time thru, anchor */
  while((x=fscanf(fp,"%s %s %s",buff,buff1,buff2))!=EOF) /* get a line */
  {  ioptr=System_Inputs;               /* points to angle */
     for(mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
     {  if((strcmp(mfptr->name,buff))==0)
    {  ifptr=(struct rule_element_type *)
                calloc(1,sizeof(struct rule_element_type));
       ruleptr->if_side=ifptr;      /* points to angle */
       ifptr->value=&mfptr->value;  /* needs address here */
       ifptr->next=(struct rule_element_type *)
                calloc(1,sizeof(struct rule_element_type));
       ifptr=ifptr->next;
       break;                       /* match found */
         }
     }
     ioptr=ioptr->next;                 /* points to velocity */
     for(mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
     {  if((strcmp(mfptr->name,buff1))==0)
    {  ifptr->value=&mfptr->value;  /* needs address here */
       break;                       /* match found */
         }
     }
     if(outptr==NULL)outptr=System_Output;/* point then stuff to output */
     for(mfptr=outptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
     {  if((strcmp(mfptr->name,buff2))==0)
        {  thenptr=(struct rule_element_type *)
                calloc(1,sizeof(struct rule_element_type));
       ruleptr->then_side=thenptr;
          thenptr->value=&mfptr->value; /* needs address here */
       break;                        /* match found */
        }
     }
     ruleptr->next=(struct rule_type *)calloc(1,sizeof(struct rule_type));
     ruleptr=ruleptr->next;
  }                                     /* END WHILE READING RULES FILE */
  close(fp);                            /* close "rules" file */
}                                       /* END INITIALIZE */
put_system_outputs()                    /* NEW */
{  struct io_type *ioptr;
   struct mf_type *mfptr;
   struct rule_type *ruleptr;
   struct rule_element_type *ifptr;
   struct rule_element_type *thenptr;
   int cnt=1;
   for(ioptr=System_Inputs;ioptr!=NULL;ioptr=ioptr->next)
   {  printf("%s: Value= %d\n",ioptr->name,ioptr->value);
      for(mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
      {  printf("  %s: Value %d Left %d Right %d\n",
           mfptr->name,mfptr->value,mfptr->point1,mfptr->point2);
      }
     printf("\n");
   }
   for(ioptr=System_Output;ioptr!=NULL;ioptr=ioptr->next)
   {  printf("%s: Value= %d\n",ioptr->name,ioptr->value);
      for(mfptr=ioptr->membership_functions;mfptr!=NULL;mfptr=mfptr->next)
      {  printf("  %s: Value %d Left %d Right %d\n",
           mfptr->name,mfptr->value,mfptr->point1,mfptr->point2);
      }
   }
/* print values pointed to by rule_type (if & then) */
   printf("\n");
   for(ruleptr=Rule_Base;ruleptr->next!=NULL;ruleptr=ruleptr->next)
   {  printf("Rule #%d:",cnt++);
      for(ifptr=ruleptr->if_side;ifptr!=NULL;ifptr=ifptr->next)
          printf("  %d",*(ifptr->value));
      for(thenptr=ruleptr->then_side;thenptr!=NULL;thenptr=thenptr->next)
          printf("  %d\n",*(thenptr->value));
   }
   printf("\n");
}                                       /* END PUT SYSTEM OUTPUTS */
get_system_inputs(input1,input2)        /* NEW */
int input1, input2;
{  struct io_type *ioptr;
   ioptr=System_Inputs;
   ioptr->value=input1;
   ioptr=ioptr->next;
   ioptr->value=input2;
}                                       /* END GET SYSTEM INPUTS */
```

**End Listing**

## Listing One *(Text begins on page 64.)*

```
//-----------------------------------------------------------------
// PPDIO Parallel Port Digital IO routines
// Version 1.0 Copyright 1993 by Brian Hook.  All Rights Reserved.
// File: PPDIO.H -- header file for the PPDIO library
// Compile with Borland C++ 3.1 -- porting to another compiler
// should be extremely trivial.
//-----------------------------------------------------------------
#ifndef __PPDIO_H
#define __PPDIO_H

//--- Pin definitions for control register -----------------------
#define PIN_1       0x01
#define PIN_14      0x02
#define PIN_16      0x04
#define PIN_17      0x08

//--- Pin definitions for status register ------------------------
#define PIN_15      0x08
#define PIN_13      0x10
#define PIN_12      0x20
#define PIN_10      0x40
#define PIN_11      0x80

//--- Interrupt enable bit definition ----------------------------
#define PTR_ENABLE_INT_BIT  0x10

//--- Function prototypes ----------------------------------------
unsigned      PPDIO_GetLptAddress( int lpt_port );
void          PPDIO_InstallISR( void interrupt (*fnc)(), int irq );
unsigned char PPDIO_ReadControlRaw( void );
unsigned char PPDIO_ReadStatusRaw( void );
unsigned char PPDIO_ReadControlCooked( void );
unsigned char PPDIO_ReadStatusRaw( void );
void          PPDIO_RemoveISR( void );
void          PPDIO_SendByte( unsigned char data );
void          PPDIO_SetBaseAddress( unsigned base_address );
void          PPDIO_SetLptPort( int lpt_port );

#endif
```

## Listing Two

```
//-----------------------------------------------------------------
// PPDIO Parallel Port Digital IO routines
// Version 1.0 Copyright 1993 by Brian Hook.  All Rights Reserved.
// File:  PPDIO.C -- code and variables for the PPDIO library
// Compile with Borland C++ 3.1 -- porting to another compiler
// should be extremely trivial.
//-----------------------------------------------------------------
#include <dos.h>
#include "ppdio.h"

static unsigned ppdio_data_register;
static unsigned ppdio_control_register;
static unsigned ppdio_status_register;
static unsigned ppdio_interrupt_no;
static unsigned ppdio_irq;

static unsigned char ppdio_old_control_value;
static unsigned char ppdio_old_8259_mask;
static void interrupt (*ppdio_old_intvec)();

unsigned PPDIO_GetLptAddress( int lpt_no )
{
unsigned far *pp = ( unsigned far * ) MK_FP( 0x40, 8 );
   //--- Assumes values of 1, 2, or 3 -----------------------------
   return ( pp[lpt_no-1] );
}
void PPDIO_InstallISR( void interrupt (*fnc)(), int irq_no )
{
static char mask[] = { 0xfa, 0xf7, 0xef, 0xdf, 0xaf, 0x7f };
unsigned char temp;

   //--- Interrupt number = IRQ no + 8 ---------------------------
   ppdio_interrupt_no = irq_no + 8;

   //--- Save original interrupt vector --------------------------
   ppdio_old_intvec = getvect( ppdio_interrupt_no );

   //--- Install new ISR ------------------------------------------
   setvect( ppdio_interrupt_no, fnc );

   //--- Enable interrupts by setting the PTR_ENABLE_INT_BIT in
   //--- the control register.  Also, OR it by 0x04 to send pin
   //--- 16 high then write out a 0 to pins 1, 14, and 17 so
   //--- that we can use the control register for input.
   ppdio_old_control_value = inportb( ppdio_control_register );
   temp = ppdio_old_control_value | PTR_ENABLE_INT_BIT | PIN_16;
   temp &= ~ ( PIN_17 | PIN_14 | PIN_1 );
   outportb( ppdio_control_register, temp );

   //--- Unmask our IRQ in the interrupt controller --------------
   ppdio_old_8259_mask = inportb( 0x21 );
   temp = ppdio_old_8259_mask & mask[ppdio_interrupt_no-10];
   outportb( 0x21, temp );

   //--- Clear pending interrupts --------------------------------
   outportb( 0x20, 0x20 );
}

unsigned char PPDIO_ReadControlCooked( void )
{
unsigned char raw_control;
unsigned char cooked_control = 0;
   raw_control = PPDIO_ReadControlRaw();

   //--- Return a control register mask that compensates for the inverse logic
   //--- of pins 1, 14, and 17, and with 0s where bits are reserved or unused.
   if ( !( raw_control & PIN_1 ) )
      cooked_control |= PIN_1;
   if ( !( raw_control & PIN_14 ) )
      cooked_control |= PIN_14;
   if ( raw_control & PIN_16 )
      cooked_control |= PIN_16;
   if ( !( raw_control & PIN_17 ) )
      cooked_control |= PIN_17;
   return ( cooked_control );
}
unsigned char PPDIO_ReadControlRaw( void )
{
   return ( inportb( ppdio_control_register ) );
}
unsigned char PPDIO_ReadStatusCooked( void )
{
unsigned char raw_status;
unsigned char cooked_status = 0;

   raw_status = PPDIO_ReadStatusRaw();
   //--- Return a status register mask that compensates for the
   //--- inverse logic of pin 11, and with 0s for any reserved or unused bits.
   if ( raw_status & PIN_15 )
      cooked_status |= PIN_15;
   if ( raw_status & PIN_13 )
      cooked_status |= PIN_13;
   if ( raw_status & PIN_12 )
      cooked_status |= PIN_12;
   if ( !( raw_status & PIN_11 ) )
      cooked_status |= PIN_11;
   return ( cooked_status );
}
unsigned char PPDIO_ReadStatusRaw( void )
{
   return ( inportb( ppdio_status_register ) );
}
void PPDIO_RemoveISR( void )
{
   //--- Restore the interrupt controller's previous state --------
   outportb( 0x21, ppdio_old_8259_mask );

   //--- Restore the original interrupt vector --------------------
   setvect( ppdio_interrupt_no, ppdio_old_intvec );

   //--- Restore the printer control register ---------------------
   outportb( ppdio_control_register, ppdio_old_control_value );
}
void PPDIO_SendByte( unsigned char data )
{
   outportb( ppdio_data_register, data );
}
void PPDIO_SetBaseAddress( unsigned base_address )
{
   ppdio_data_register    = base_address;
   ppdio_status_register  = base_address + 1;
   ppdio_control_register = base_address + 2;
}
void PPDIO_SetLptPort( int lpt_port )
{
   PPDIO_SetBaseAddress( PPDIO_GetLptAddress( lpt_port ) );
}
```

## Listing Three

```
//-----------------------------------------------------------------
// PPDIO Parallel Port Digital IO routines
// Version 1.0 Copyright 1993 by Brian Hook.  All Rights Reserved.
// File:  DIO.C -- this is an example how you could use the PPDIO
// routines. This could be used as a framework upon which you
// could build real applications.
// Compile with Borland C++ 3.1 -- porting to another compiler
// should be extremely trivial.
//-----------------------------------------------------------------
#include <conio.h>
#include "ppdio.h"

volatile int isr_called = 0;
void huge interrupt MyISR( void )
{
   isr_called = 1;

   //--- Normally you would read the input pins here and do something important
   //--- Signal end of interrupt to the interrupt controller ------
   outportb( 0x20, 0x20 );
}
void main( void )
{
   //--- Use LPT1 -------------------------------------------------
   PPDIO_SetLptPort( 1 );

   //--- Install our ISR on IRQ 5 ---------------------------------
   PPDIO_InstallISR( MyISR, 5 );

   //--- Run until either a key is pressed or interrupt is generated on IRQ 5
   while ( !kbhit() && !isr_called ) {
   }
   PPDIO_RemoveISR();
}
```

**End Listings**

## Listing One *(Text begins on page 92.)*

```
//File: GRPHPHEN.H
#ifndef __GRPHPHEN_H
#define __GRPHPHEN_H

//Header for EOS class representing a phenotype.
//You need EOS v1.1 to compile this code
#ifndef __PHENO_H
#include "pheno.h"
#endif //__PHENO_H

class CGraphDrawingPheno : public TPhenotype
{
   public:
    CGraphDrawingPheno(CGAGraphDriver &driver,int width, int height) ;
    ~CGraphDrawingPheno() ;
    double CalcFitness() ;
    void Decode(PTGenotype geno) ;
    PTPhenotype Copy()  ;
    void GetPhenoInfo(void *pInfoStruct)  ;
    void GetNearestEmptyCell(const int row, const int col, int &actualRow,
                                                    int &actualCol) ;
    BOOL Adjacent(WORD node1, WORD node2) ;
    BOOL Diagonal(WORD node1, WORD node2) ;
    BOOL FindNode(const WORD node, int &row, int &col) ;
    double Distance(WORD node1, WORD node2) ;
    double RectDistance(WORD node1, WORD node2) ;
private:
    int m_Width  ;
    int m_Height ;
    CWordMatrix *m_pGrid ;      //grid where each entry is a node
                                //                  number or EMPTY_CELL
    CGAGraphDriver &m_Driver ;  //interface to the graph driver class
    int * m_GridIndex[2] ;      //index into grid to quickly locate nodes
};
```

## Listing Two

```
//File: GRPHPHEN.CPP
#include "stdafx.h"
//eos headers
#include "eos.h"
#include "eosutil.h"
#include "geno.h"

//graph GA headers
#include "grphphen.h"
#include "wmatrix.h"
#include "gdriver.h"
#include "grphutil.h"

const HIGHEST_REWARD = 10 ;
const MEDIUM_REWARD = 5 ;
const SMALLEST_REWARD = 1 ;
const HIGHEST_PENALTY = 10 ;
const MEDIUM_PENALTY = 5;
const SMALLEST_PENALTY = 1;

CGraphDrawingPheno::CGraphDrawingPheno(CGAGraphDriver &driver, int width,
                                                            int height)
    : m_Driver(driver)
{
    m_Width = width ;
    m_Height = height ;
    m_pGrid = new CWordMatrix(height,width,EMPTY_CELL) ;
    m_GridIndex[0] = new int [m_Driver.GetNumNodes()];
    m_GridIndex[1] = new int [m_Driver.GetNumNodes()];
}
CGraphDrawingPheno::~CGraphDrawingPheno()
{
    delete m_pGrid ;
    delete [] m_GridIndex[0] ;
    delete [] m_GridIndex[1] ;
}
double CGraphDrawingPheno::CalcFitness()
{
    WORD numNodes = (WORD) m_Driver.GetNumNodes() ;
    long maxDist = (m_Width + m_Height) ;
    maxDist*=maxDist;
    //set base fitness so even the worst case phenotype
        //          will not bring fitness below 0
    int connectivity = m_Driver.GetConnectivity() ;
    double base_fitness = numNodes*(numNodes-1) * maxDist ;
                                                    //* connectivity;
    double fitness = base_fitness ;
    for (WORD node1=0;node1<numNodes;node1++) {
        int node1Connections=Max(m_Driver.GetNumConnections(node1),1);
        for (WORD node2=0;node2<numNodes;node2++) {
            if (node1 == node2)
                continue ;
            BOOL bConnected = m_Driver.Connected(node1,node2) ;
            int node2Connections =
                                Max(m_Driver.GetNumConnections(node2),1);
            double distance = Distance(node1,node2) ;
            distance*=distance;
            if (bConnected && distance > 4)  {
                fitness -= distance ;
                                  //(node1Connections+node2Connections) ;
                continue ;
                }
            if (!bConnected && distance <= 4)  {
                fitness -= 4/distance ;
                                   //(node1Connections+node2Connections) ;
                continue ;
                }
            }
        }
    ASSERT(fitness >= 0);
    return fitness ;
}
void CGraphDrawingPheno::Decode(PTGenotype pGeno)
{
    WORD numNodes = (WORD) m_Driver.GetNumNodes() ;
    int rowAlleleLen =  m_Driver.CalcRowAlleleLength() ;
    int colAlleleLen =  m_Driver.CalcColAlleleLength() ;
    int offset = 0 ;
    for (WORD node=0;node<numNodes;node++) {
        char rowAllele[16], colAllele[16] ;
                          //we know that these are no bigger than sizeof(WORD)
        for(int bit=0;bit<rowAlleleLen;bit++)
            rowAllele[bit] =
                                  pGeno->GetExpressedGeneValue(offset++,0) ;
        for(bit=0;bit<colAlleleLen;bit++)
            colAllele[bit] =
                                  pGeno->GetExpressedGeneValue(offset++,0) ;
        int codedRow = AllelesToInt(rowAllele,0, rowAlleleLen-1) ;
        int codedCol = AllelesToInt(colAllele,0, colAlleleLen-1) ;
        int actualRow, actualCol ;
        GetNearestEmptyCell(codedRow,codedCol,actualRow,actualCol) ;
        m_pGrid->SetAt(actualRow, actualCol, node) ;
        m_GridIndex[0][node] = actualRow ;
        m_GridIndex[1][node] = actualCol ;
        }
}
PTPhenotype CGraphDrawingPheno::Copy()
{
    CGraphDrawingPheno * pPheno =
                        new CGraphDrawingPheno(m_Driver,m_Height,m_Width) ;
    return pPheno ;
       //don't copy values because these are derived by the genotype via Decode
}
void CGraphDrawingPheno::GetPhenoInfo(void *pInfoStruct)
{
    *((CWordMatrix **)pInfoStruct) = m_pGrid ;
}
//Algorithm resolves collisions by searching around the neighborhood of
// (row,col) in the grid for an empty cell. The row and col of the empty cell
// is returned in actualRow and actualCol.
void CGraphDrawingPheno::GetNearestEmptyCell(const int row, const int col,
                                            int &actualRow,int &actualCol)
{
    //insure we are in range!
    actualRow = row % m_Height ;
    actualCol = col % m_Width ;
    //if we find and empty cell then no search necessary
    if (m_pGrid->GetAt(actualRow,actualCol) == EMPTY_CELL)
        return ;
    else { //search for "nearest" empty cell
        int maxDist=Max(m_Height,m_Width) ;
        int actualRow2 = actualRow ; //save actuals
        int actualCol2 = actualCol ;
         //start at a distance of 1 and search outward
        for (int dist=1;dist<maxDist;dist++) {
            //First check "sides"
            for(int i=-dist; i<=dist;i++) {
                for(int j=-dist;j<=dist;j++) {
                    if (i!=j && (j==dist || j==-dist ||
                                            i==dist || i==-dist)) {
                    actualCol = actualCol2+j ;
                    actualRow = actualRow2+i ;
                    if(actualCol >= 0 && actualCol
                                                          < m_Width &&
                       actualRow >= 0 && actualRow
                                                          < m_Height &&
                m_pGrid->GetAt(actualRow,actualCol) == EMPTY_CELL)
                        return ;
                    } //if
                } // for j
            } //for i
            //Now check 4 corner cells
            actualCol = actualCol2+dist ;
            actualRow = actualRow2+dist ;
            if(actualCol < m_Width &&
                actualRow < m_Height &&
                    m_pGrid->GetAt(actualRow,actualCol) ==
                                                            EMPTY_CELL)
                return ;
            actualCol = actualCol2-dist ;
            actualRow = actualRow2+dist ;
            if(actualCol >= 0 &&
                actualRow < m_Height &&
                m_pGrid->GetAt(actualRow,actualCol) ==
                                                            EMPTY_CELL)
                return ;
            actualCol = actualCol2+dist ;
            actualRow = actualRow2-dist ;
            if(actualCol < m_Width &&
                actualRow >= 0 &&
                m_pGrid->GetAt(actualRow,actualCol) ==
                                                            EMPTY_CELL)
                return ;
            actualCol = actualCol2-dist ;
            actualRow = actualRow2-dist ;
            if(actualCol >= 0 &&
                actualRow >= 0 &&
                m_pGrid->GetAt(actualRow,actualCol) ==
                                                            EMPTY_CELL)
                return ;
            } //for dist
        } //else
    return ;
}
//Return TRUE if node1 is adjacent to node2 on the grid
BOOL CGraphDrawingPheno::Adjacent(WORD node1, WORD node2)
{
    int row1, col1 ;
    if (!FindNode(node1,row1,col1))
        return FALSE ;
    int row2, col2 ;
    //look up
```

```
    row2=row1-1 ;
    if (row2 >= 0 && m_pGrid->GetAt(row2,col1) == node2)
        return TRUE ;
    //look down
    row2=row1+1 ;
    if (row2 < m_Height && m_pGrid->GetAt(row2,col1) == node2)
        return TRUE ;
    //look left
    col2=col1-1 ;
    if (col2 >= 0 && m_pGrid->GetAt(row1,col2) == node2)
        return TRUE ;
    //look right
    col2=col1+1 ;
    if (col2 < m_Width && m_pGrid->GetAt(row1,col2) == node2)
        return TRUE ;
    return FALSE ;
}
//Return TRUE if node1 is diagonal to node2 on the grid
BOOL CGraphDrawingPheno::Diagonal(WORD node1, WORD node2)
{
    int row1, col1 ;
    if (!FindNode(node1,row1,col1))
        return FALSE ;
    int row2, col2 ;
    //look upper left
    row2=row1-1 ;
    col2=col1-1 ;
    if (row2 >= 0 && col2 >= 0 && m_pGrid->GetAt(row2,col2) == node2)
        return TRUE ;
    //look lower left
    row2=row1+1 ;
    col2=col1-1 ;
    if (row2 < m_Height && col2 >= 0 && m_pGrid->GetAt(row2,col1) == node2)
        return TRUE ;
    //look lower right
    row2=row1+1 ;
    col2=col1+1 ;
    if (row2 < m_Height && col2 < m_Width && m_pGrid->GetAt(row1,col2) ==
                                                                     node2)
        return TRUE ;
    //look upper left
    row2=row1-1 ;
    col2=col1+1 ;
    if (row2 >= 0 && col2 < m_Width && m_pGrid->GetAt(row1,col2) == node2)
        return TRUE ;
    return FALSE ;
}
//Return the Euclidean distance between nodes on the grid
double CGraphDrawingPheno::Distance(WORD node1, WORD node2)
{
    int row1, col1, row2, col2 ;
    if (FindNode(node1,row1,col1) && FindNode(node2,row2,col2)) {
        double diffRow = row1 - row2 ;
        double diffCol = col1 - col2 ;
        return sqrt(diffRow*diffRow + diffCol*diffCol) ;
        }
    else
        return sqrt(m_Height*m_Height + m_Width*m_Width) ;
}
//Return the recti-linear distance between nodes on the grid
double CGraphDrawingPheno::RectDistance(WORD node1, WORD node2)
{
    int row1, col1, row2, col2 ;
    if (FindNode(node1,row1,col1) && FindNode(node2,row2,col2)) {
        double diffRow = row1 - row2 ;
        double diffCol = col1 - col2 ;
        return Abs(diffRow) + Abs(diffCol) ;
        }
    else
        return m_Height + m_Width ; //really an error ?!?
}
//Use an index to quickly locate a node on the grid
BOOL CGraphDrawingPheno::FindNode(const WORD node, int &row, int &col)
{
    if (node >= m_Driver.GetNumNodes())
        return FALSE ;
    row = m_GridIndex[0][node] ;
    col = m_GridIndex[1][node] ;
    return TRUE ;
}
```

## Listing Three

```
//File: GDRIVER.H
#ifndef __GDRIVER_H__
#define __GDRIVER_H__
//flag an empty cell in the grid
const EMPTY_CELL = 0xFFFF ;

class CGAGraphDriver
{
    //Interface
public:
    CGAGraphDriver(int numNodes, int width, int height) ;
    ~CGAGraphDriver() ;
    void SetGraph(CWordMatrix &graph) ;
    void Optimize(int numGenrations) ;
    void DrawOptimized(CDC &dc) ;
    void DrawUnOptimized(CDC &dc) ;
    //Query members (const)
    //Calc the length of a chromosome
    //needed based on the graph and grid
    UINT CalcChromosomeLength() const ;
    UINT CalcRowAlleleLength() const ; ;
    UINT CalcColAlleleLength() const ; ;
    int GetWidth() const ;
    int GetHeight() const ;
    int GetNumNodes() const ;
    BOOL Connected(WORD node1, WORD node2)  const;
    int GetNumConnections(WORD node) const ;
    int GetConnectivity() ;
    void Stop() ;
    PTIndividual m_pBest ;
    PTIndividual m_pWorst ;
    BOOL m_Stop ;
    //Implementation
private:
    //Draw the graph in this grid
    void Draw(CDC &dc, CWordMatrix &Grid) ;
    //num nodes in the graph
    int m_NumGraphNodes ;
    //width of grid to draw on (in cells)
    int m_GridWidth ;
    //height of grid to draw on (in cells)
    int m_GridHeight ;
    //connection table representation of a graph
    CWordMatrix *m_pGraph ;
    //GA that will find the "optimal" drawing
    //of the graph on the grid
    TBasicGA  *m_pTheGA ;
} ;
```

## Listing Four

```
//File: GDRIVER.CPP
//Used as an interface class to the GA.
//Stores the representation of the graph as
//a connection grid.

//required headers
#include "stdafx.h"

//Headers needed for EOS programs
//You need EOS v1.1 to compile this code
#include "eos.h"
#include "eosutil.h"
#include "geno.h"
#include "individ.h"
#include "gaenviro.h"

//headers specific to graph GA
#include "wmatrix.h"
#include "gdriver.h"
#include "grphutil.h"
#include "graphga.h"

//GA parameters used, these need not be
//hard coded in advanced implementations
const int POP_SIZE = 20 ;
const double PX = 0.7 ;
const double PM = 0.03 ;
const double RAND_SEED=0.76451 ;

//DRAWING parameters used, these need not be
//hard coded in advanced implementations
const int CELL_WIDTH = 30 ;
const int CELL_HEIGHT = 30  ;
const int CELL_SPACE = 30 ;

//Driver constructor initializes a graph with  numNodes and a
//grid that the graph will be optimized to draw on (width x height)
CGAGraphDriver::CGAGraphDriver(int numNodes, int width, int height)
{
    m_NumGraphNodes = numNodes;
    m_GridWidth = width ;
    m_GridHeight = height ;
    //graph represented as boolean connection matrix
    m_pGraph = new CWordMatrix(m_NumGraphNodes,m_NumGraphNodes) ;
    //The Graph GA object
    m_pTheGA = new CGraphDrawerGA(*this) ;
    m_pBest = NULL ;
    m_pWorst = NULL ;
    m_Stop = FALSE ;
}
//Clean up in the destructor
CGAGraphDriver::~CGAGraphDriver()
{
    delete m_pGraph ;
    delete m_pTheGA ;
}
//set the conections from graph into the member m_pGraph
void CGAGraphDriver::SetGraph(CWordMatrix &graph)
{
    for (int row = 0 ; row < m_NumGraphNodes; row++)
        for (int col = 0 ; col < m_NumGraphNodes; col++)
            m_pGraph->SetAt(row,col,graph[row][col]) ;
}
// Optimize the drawing of the graph by first initializing the GA's population
// and environment. Then execute the GA for numGenerations generations
void CGAGraphDriver::Optimize(int numGenerations)
{
    m_pTheGA->CreatePopulation(POP_SIZE) ;
    m_pTheGA->CreateEnvironment(PX,PM,RAND_SEED) ;
    m_pTheGA->Evolve(numGenerations) ;
}
//Draw the optimized graph on the Windows DC
void CGAGraphDriver::DrawOptimized(CDC &dc)
{
    CWordMatrix *pGrid ;
    m_pBest->GetPhenoInfo(&pGrid) ;
    Draw(dc,*pGrid) ;
}
//Draw the un-optimized graph on the Windows DC
void CGAGraphDriver::DrawUnOptimized(CDC &dc)
{
    CWordMatrix *pGrid ;
    m_pWorst->GetPhenoInfo(&pGrid) ;
    Draw(dc,*pGrid) ;
}
```

*(continued on page 147)*

# A Little RISC Lands Apple in the Soup

Michael Swaine

In the early 1980s, the British microcomputer market was dominated by British companies, primarily Sinclair and Acorn.

It was an unlikely scenario.

The microcomputer revolution was by this time becoming institutionalized. What, only a few years before, had been a marginal market of electronics hobbyists selling to other electronics hobbyists had become a venture-capital-attracting international industry. IBM had come in and legitimized the industry, was the commonly heard—and true, even if incomplete—explanation.

All the early shots in this revolution had been fired in the United States, and all the big companies—no surprise—were U.S. companies, some of which had established manufacturing facilities in Europe. The European market, taken as a whole, was only a fraction of the U.S. market. The British market was a fraction of that fraction, and, unlike some European countries, Britain didn't have high tariffs to keep out American computers. By all logic, American computer companies should have been able to walk all over the homegrown brands.

But that's not what happened. British computer companies were bucking the odds and winning. What was going on?

### Who Were These Guys?

One of the things that stands out when you look at the British microcomputer scene in those days is the Cambridge connection. Sinclair and Acorn had Cambridge University connections in common, and Acorn in particular maintained close ties with the university, drawing on it for personnel, ideas, and support. Cambridge may have been one strength of these companies.

But Sinclair and Acorn differed in many ways. For one thing, Clive Sinclair went for the high-concept products: The World's Cheapest Computer, The First Practical Electric Car. The Acorn crew were less flamboyant. They just built a computer.

The Sinclair computer was one of the first users of the Zilog Z80, arguably the first microprocessor created specifically to be the CPU of a personal computer. Arguably. The Acorn used a chip originally intended for controller use: the Rockwell 6502. The Acorn developers got to be experts in the 6502, just as Apple cofounder Steve Wozniak did.

Clive Sinclair, like Nolan Bushnell in the United States, founded several companies, explored diverse industries, and had flashes of high visibility; Sinclair, though, has been off American radar for years. The Acorn team prospered with less abrupt ups and downs and has significant visibility today. It was the BBC deal that made their fortune.

The British Broadcasting Company had decided to launch a computer-education television show that would run throughout the UK, and it wanted a BBC microcomputer to sell to viewers of the show. It was a savvy plan, and when Acorn got the BBC contract, both Acorn and the BBC thought that they could sell over ten thousand computers despite the small size of the nascent British market.

To date, Acorn has sold nearly two million BBC Micro-compatibles, and the company has grown from a typical microcomputer company of the early '80s with a staff of a couple dozen to a multimillion-pound company with hundreds of employees.

When it came time, in the mid-1980s, to admit that the 6502 had had its day, the Acorn guys did something telling. Rather than accept the conventional wisdom about the "right" microprocessor for the next generation of computers, they fell back on their expertise, or perhaps just their old habits. They designed their own.

What they came up with was the kind of chip you might expect old 6502 hackers to design: a small instruction set, low power consumption, small die size, potentially low cost. It may have been of only academic interest to them that these are now the characteristics of low-end RISC chips. They weren't trying to develop the first commercial RISC processor. They just wanted a better 6502.

What they came up with was the Acorn RISC Machine, or ARM. The first ARM chip was shown fully functional in April of 1985. It operated reliably at 8 MHz, although designed to operated with a 4-MHz clock. It was a $3\mu$ device of about 25,000 transistors. Initially, the ARM1 was offered as a coprocessor in the BBC computer. The second generation ARM2 was used by Radius in one of its first graphics accelerator cards for the Macintosh. The ARM2 also saw service in the movies, being used in the robotic controller from MicroRobotics of Cambridge, England, that controlled the robot turtles in the movie *Teenage Mutant Ninja Turtles.*

### Meanwhile, Back in the Colonies...

Apple formed its Advanced Technology Group (ATG) in 1986. At that time Acorn, facing competitive pressures from clones, had just been acquired by Olivetti and was soon to release its first ARM-based computer, the Archimedes, to a lukewarm response. Apple's ATG was chartered to explore new technologies that could be of use to Apple in the '90s. One technology that ATG evaluated and took note of for possible inclusion in Apple products was Acorn's ARM processor, but nothing was done with the ARM at the time.

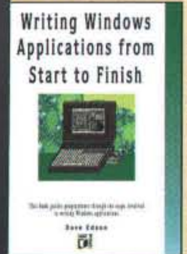
Somewhat later, a skunkworks within ATG called the Advanced Products Group (APG) took on the mission of developing a new system architecture that they were calling Newton. The trip to Newton had a lot of side trips and blind alleys. It was apparently Michael Chao's Knowledge Navigator pitch to John Sculley that tipped the balance from a tablet form factor to the handheld device that Apple eventually released.

One of the other alleys explored involved the microprocessor. For some time the AT&T Hobbit chip was considered. What they were looking for was a processor with characteristics that sounded like those of a microcontroller rather than a computer CPU: small die size, low cost, low power consumption, instruction set efficiency, ease of embedding in ASIC designs. In 1990, RISC

*(continued on page 112)*

# noble bookstar bookstop";
# get next bookThere();
# }

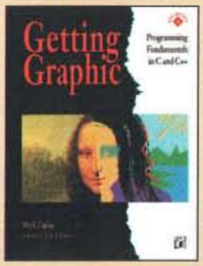
## WINDOWS PROGRAMMING




### Writing Windows Applications from Start to Finish
*by Dave Edson*

Using a real-world example, this book guides programmers through the steps involved in writing Windows applications, including user interface design, file content, data structures, coding, testing and documentation. Readers not only learn how to plan and design an application from start to finish but also acquire a home inventory program. Disk contains complete source code. (BK/DISK)

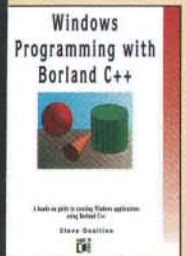
**List Price $39.95**




### Getting Graphic: Programming Fundamentals in C and C++
*by Mark Finlay*

This book teaches the fundamentals of graphics programming. It shows C and C++ programmers how to plot points on a screen, draw geometric shapes, design 3-D figures and more. This book/disk package is filled with sophisticated and usable source code examples and sample graphic images. *Getting Graphic* is a perfect introduction to the exciting world of graphics. (BK/DISK)

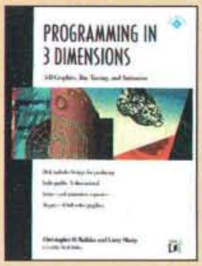
**List Price $39.95**




### Windows Programming with Borland C++
*by Steve Oualline*

Learn to create Windows applications using Borland C++ 3.1. This hands-on guide shows you how. It leads you through the design, implementation and debugging process used to create a program. You'll learn how to create various Windows applications including a Find File program and a fish screen saver. Complete with source code disk. (BK/DISK)

**List Price $39.95**




### Programming in 3 Dimensions
*by Christopher D. Watkins and Larry Sharp*

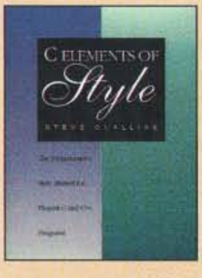
Required reading! This one is for all computer graphics enthusiasts who want a detailed look at 3-D graphics and modeling. Also features discussions of popular ray tracing methods and computer animation. Includes eight pages of full-color graphics. Provides C source code and numerous examples. MS/PC-DOS disk contains sample source code. A must have! (BK/DISK)

**List Price $39.95**


### Borland Pascal with Objects 7.0
*by José de Jesús and Abad Godoy*

*Borland Pascal with Objects 7.0* integrates DOS and Windows like no other development tool on the market. This book will teach you object-oriented programming and how to incorporate existing objects into your Pascal programs. Structured as a tutorial, this book will guide you from the basics of Pascal for DOS, through Turbo Vision and up to ObjectWindows. If you're a beginner, you can use this book to build your expertise from the bottom up. If you're an experienced programmer, you can leaf through familiar areas into more advanced topics. (BK/DISK)

**List Price $39.95**




### C Elements of Style
*by Steve Oualline*

This concise guide covers the rules of good program design, teaching C and C++ programmers how to write programs that can be easily read, understood and maintained by others. Whether you're a student or professional programmer, you'll benefit from the many tips and techniques for constructing elegant, reliable code.

**List Price $21.95**


### Plug & Play Programming
*by William Wong*

This book introduces a new programming technique based on an object-oriented class called "plugs." It teaches programmers how to use plugs to design programs with interchangeable components that can easily be ported from one program to another without modification. Includes MS/PC DOS disk containing ready-to-use libraries of plugs plus source code. (BK/DISK)

**List Price $39.95**


### C++ Database Development
*by Al Stevens*

Al Stevens, the popular *Dr. Dobbs* columnist, provides all the tools you need to explore the full potential of the C++ object-oriented language. Learn to use C++ to design and develop utilities and database management programs. Topics include database fundamentals and design, database management, building the software and much more.

**List Price $24.95**

Call 1-800-344-2470 for the location nearest you.

**Barnes & Noble | BOOKSTAR | BOOKSTOP**

CIRCLE NO. 845 ON READER SERVICE CARD

*(continued from page 109)*
looked promising, and ARM looked particularly good.

To ensure that future ARM processors would fit Apple's evolving needs, Apple made a deal. It was an early example of the joint ventures that Apple continues to pursue today. Apple UK joined forces with Acorn and VLSI Technology, with whom Acorn had worked in producing the first ARM chips, to form ARM Ltd.

ARM Ltd.'s ARM 610 became the processor for the first Newton devices, the Apple MessagePad and Sharp ExpertPad. (ARM6 devices like the ARM 610 really represent the fourth generation of ARM devices; apparently the numbering skipped 4 and 5.)

ARM was on a roll. In 1992, 3DO announced that the ARM60 would be used in its Interactive Multiplayer. ARM6 devices are also seeing use in controller applications, such as fuzzy-logic controllers.

The ARM6 family embodies full 32-bit addressing and support for both Big-endianness and Little-endianness, a requirement imposed by Apple. The ARM610 includes a 4-Kbyte cache, a write buffer, and a MMU, all in a package smaller than a 386. The MMU implements memory domains and permissions designed to provide hardware support for modern operating-system memory-management strategies like multilevel memory protection, memory paging, demand-paged virtual memory, and object-oriented memory with background garbage collection. The last of these turns out to be crucial to the Newton model for object storage.

The rest of this column looks at some of the characteristics of that model.

### A Little Selfishness

Newton's model of object-oriented technology is reported to be related to SELF, an object-oriented dynamic language developed by Smith and Unger at Stanford University about the time the ARM1 chip was seeing first silicon. NewtonScript is not SELF, though, or Dylan, or any other language. It has some unique characteristics.

One characteristic that NewtonScript does share with SELF is the "everything is an object" approach. The SELF model is unusual among object-oriented languages in that it isn't built around classes. The slogan "everything is an object" means that objects inherit directly from other "prototype" objects, as distinct from the more familiar class-based inheritance.

Newton's object-oriented language, NewtonScript, diverges from SELF in many ways, but has much the same spirit. It has prototype inheritance, as well as "parent" inheritance. But not everything is an object to NewtonScript. Chunks of data that can fit into 32 bits (integers, characters, Boolean values) are addressed via immediate reference, while everything else is a pointer reference. All these pointer-referenced data are stored in the heap as, yes, objects. Some object-data types are: symbols, reals, arrays, strings, and frames. The most important type of object in the Newton object-storage model is the frame.

A frame is a data structure containing named references to objects of arbitrary data type. It's much like a *struct* or record in other languages. A frame can also contain functions.

Example 1 is a typical NewtonScript frame. Frames in NewtonScript are delimited by braces ({}). The named data items within a frame are called "slots." Each slot is specified by its name, a colon, and its value. The slots are separated from one another by commas. Example 1 shows a *_proto* slot (more about this shortly), an integer constant slot, a Boolean constant slot, a string constant slot, a function slot (this is how methods are implemented in NewtonScript), and a slot that is itself a frame.

The *_proto* slot indicates one of the modes of inheritance, prototype inheritance. To establish that frame 2 inherits in this way from frame 1, you give frame 2 a *_proto* slot and give that slot a reference to frame 1 as its value. Frame 1 is then frame 2's prototype. Frame 2 can use (inherit) slots of frame 1, can override them with its own slot declarations, and can have additional slots that frame 1 doesn't have. Since functions can appear in frame slots, functions can also be overridden and inherited in this same way.

By the way, to send that method *exampleFunction* as a message to the frame *exampleFrame*, the syntax is *exampleFrame : exampleFunction.*

A couple of points will indicate how you work with this kind of inheritance: Inheritance is by reference, and prototypes can be in ROM. The implication is that if there is any chance that a reference to a certain slot may be a reference to ROM, you should declare that slot in frame 2, even though it is declared in frame 1 and inherited from it.

In fact, the whole Newton user interface essentially resides in prototypes in ROM, and you can use them as the prototypes for components of your applications. Simple Newton applications can be developed without any actual coding by using visual programming tools in the Newton Toolkit (NTK). These tools mainly facilitate this process of using ROM prototypes as the prototypes for components of your application. More complex applications will require some actual coding, of course, and it should be noted that only the user-interface elements can be used in this way. The rest of your app has to be built the hard way.

```
exampleFrame := {
      _proto: protoFrame,
      index: 1,
      active: TRUE,
      name: "Name of Frame",
      exampleFunction:
            func(param)
            begin
                  return param * 10;
            end
      otherFrame:
            {     owner: "Mike Swaine",
                  ownerAddress: "72511,172"
}
      } ;
```

***Example 1:*** *A NewtonScript frame.*

### Look for the Union Label

To understand how Newton stores object data, you need to know about stores, soups, and entries.

Newton objects can, at least for the current devices, reside in one of two places: in memory (ROM or RAM) or on a PCMCIA card. The memory and the card are called "stores." Other stores may be available on future Newton devices.

Stores contain collections of data called "soups." All the data in a store are in soups, and a store can hold many soups. If a store is like a volume, a soup is like a database on the volume.

Soups are made up of "entries." An entry is a frame. If a soup is like a database, an entry is like a record.

This model—physical stores containing soups made up of entries, and entries that are *struct*-like frames of object data—shows that Newton objects basically reside on Newton's physical storage devices, but it creates a false impression.

Because it isn't the simple soups that matter most in Newton software development, but cross-store collections called "union soups." Union soups seamlessly merge data from soups of different stores. If programmers use union soups rather than soups, then users can always decide where they want their data stored. In a machine with less than 200K of user-available RAM, you can be sure that's an issue. The moral for Newton developers: Use union soups.

Naturally, there's an exception to this rule. Preferences are stored in the System soup in ROM only. Every application adds at least one entry to this soup, which is not a union soup.

All existing soups (the "names" soup used by the bundled Names application,

for example) are available to your application, and you are encouraged to use them. You can add your own data to these existing soups by adding a slot. To avoid conflicts, Apple encourages you to add just one slot, using your *appSymbol* as the name entry for the slot.

Note the distinction: Adding an entry to a soup is like adding a record to a database. Adding a slot is like adding a field.

**Soup Management**

Besides automatic garbage collection, Newton provides a lot of built-in data management. Soups automatically maintain indexes of their entries. You specify these indexes when you create a soup, but indexes can be added and removed dynamically. Currently, the only kind of index supported is "slot," but future versions of NewtonScript may support others. Using a slot index means that the index key is the value of a particular slot that appears in each entry.

The function *theStore : createSoup ( soupNameString, indexArray )* creates a soup of the specified name in the store named *theStore. IndexArray* is a frame describing the initial index(es) you are creating for the store. You don't have to create any, since indexes can be added later. Soups can contain any mishmash of entries, but unless all entries have at least one slot in common, it won't be possible to specify an index that lets you search the whole soup. Some points on managing soup entries: When you add an entry to a soup, you actually add the transitive closure of the entry. Altering an entry doesn't update the store; you need to call *EntryChange*. The Newton operating system calls *EntryChange* every so often when idle, but applications will typically have to know when to call *EntryChange* themselves. The only way to get at the entries in a soup is via a "query." A query can use an index, or some other kind of search, like searching all string slots in all entries for a specified search string. A query returns a set of entries, and these entries are then accessed through an object called a "cursor."

A cursor is a pointer to one of the entries in this returned set. The cursor is advanced to the next entry in the set or otherwise repositioned by sending it messages.

The Newton approach to handling persistent-object data has some distinctive and, I think, interesting characteristics. I suspect I'll have more to say about it in future columns.

**DDJ**

To vote for your favorite article, circle inquiry no. 11.

## Essential Chart for Windows

*The most powerful and easiest way to develop charts for Windows*

A graphical presentation of data adds important appeal and utility to any application. Essential Chart makes your data come alive in full color 2D or 3D, static or real-time charts and graphs. This comprehensive Windows DLL adds extensive functionality including true 3D rotation and perspective, and the ability to scale text using built-in True-Type font support. Plus, a powerful, time-saving CASE tool allows you to interactively create and design charts and chart templates without programming.

### Features:

- 18 different chart types
- 2D and 3D bar, pie, pyramid, ribbon and area charts
- Financial and analysis chart types
- True 3D perspectives and rotations
- Real-time charting with live data
- Supports True-Type fonts for text scaling and resizing
- Windows printer support
- Displays chart in a parent or child window
- Source available

**Pricing:** $399

## object-Menu

*The cross–platform application framework for C++*

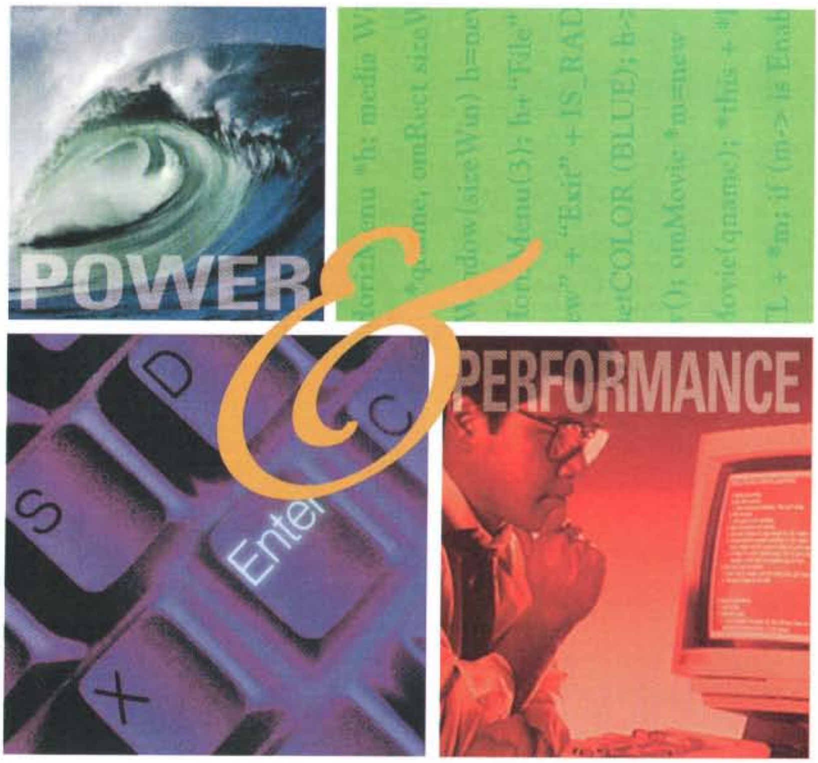
A good interface design can be the deciding factor in today's competitive marketplace; object-Menu makes it happen in several ways. Built-in aesthetics make it easy to create interfaced styling such as Windows, Motif, or your own custom design. Flexible interface configuration allows you to mold the interface to the application, not the other way around. And, portability to DOS, Windows/NT and OS/2 enable you to offer your product to multiple target markets with a single engineering effort. Additionally, object-Menu's intuitive architecture, straightforward methodology and Visual Design tool actually speed GUI development to allow more time to focus on your application.

### Features:

- Currently supports DOS graphics, Microsoft Windows, Windows NT and OS/2.
- Visual Design Tool included with automatic code generation
- Extensive "behind the scenes support" keeps your programs small and easier to debug and maintain
- Comprehensive set of interface and graphics objects including multi-media objects
- Supports multiple browsers or child windows off the main application
- Extensive data entry capability including real time field formatting and validation as well as unlimited data types
- Hypertext help system with help authoring plus hypertext primitives
- Intermix graphical icons with menus, buttons, text
- Includes the MetaWindow/XL high performance graphics library
- Supports major C++ compilers and all popular DOS graphics libraries

Pricing:

| | |
|---|---|
| Single platform (choose DOS, Windows, NT or OS/2) | $299 |
| Single platform with source | $449 |
| object-Menu Professional (DOS, Windows/NT with source) | $699 |
| object-Menu Professional-OS/2 (DOS, Windows/NT, OS/2 with source) | $899 |




LIFEBOAT®
PUBLISHING

1163 Shrewsbury Ave. • Shrewsbury, NJ 07702
201-762-6965 • Fax 908-389-9227 • BBS 201-762-0339

**CALL to ORDER**
**800-447-1955**

# Borland Nonsense: Ready, Aim, Shoot!

Al Stevens

The marksman: Borland. The target: Borland's foot. The weapon: Borland C++ 4.0's No-Nonsense License Statement.

Read the saga of how a company, known far and wide as the software developer's friend, dropped their guard, let their lawyers rewrite their no-nonsense license statement, and plugged themselves squarely in the pedal extremity.

Our story begins with the patent insanity. Unbeknownst to us, Borland holds a patent on their VROOMM overlay technology, and they have several other software patents pending. Those patents, when granted, will cover algorithms that are implemented within their libraries, DLLs, database engines, and other redistributable modules. In theory, when you build a program with their compiler, the executable code will contain algorithms covered by a Borland patent. Setting aside the question of the validity of software patents in general, the result is that you are distributing a program made with patented components. By law, you need a license from Borland to distribute those components.

Licenses can be obtained in many ways. You can pay a one-time fee for an unlimited license. You can pay a per-copy royalty. The holder can give you a royalty-free license. You can exchange patent licenses. Or you can be denied the license. If the patent holder does not want any competition, or does not want you in business for some reason, they can refuse to grant you a license. You would need to find another way to write your program.

Traditionally, Borland and other compiler vendors include this grant in the license conditions with which you tacitly agree when you break the seal and use the product.

### The Borland Dilemma

Prior to version 4.0, Borland's C++ no-nonsense license statement made no mention of patents. It granted to each registered user a license to distribute compiled programs without additional fees being charged. But someone at Borland saw something wrong with that. They reasoned that a major competitor could use Borland technology to build competing tools and applications.

As the self-professed dominant vendor of tools and applications, Borland found itself facing an internal conflict of agendas. The languages division wants to provide software developers with the best software development technology. The applications folks want to maintain dominance in a marketplace where competitors can use those superior Borland tools.

As one Borland spokesman put it, Microsoft could buy one copy of Turbo C++ for $99.00 and receive unlimited use of the patented VROOMM technology in applications that would then compete with Borland applications. Borland wanted to keep the competition from using its patented technology against it and continue at the same time to be responsive to the needs of its language customers. It was the old cliché about having your cake and eating it, too, which is what Borland tried to do. But what it came up with was met by an overwhelming firestorm of user reaction.

What lit the fire? Well, in times past, you could distribute as many copies of programs as you wanted. Under the terms of the new no-nonsense license statement, you could distribute only up to 10,000 copies per year of your Borland-compiled application. To distribute more copies than that, you would have to get Borland's permission. The reasoning behind this peculiar condition, as spokespeople explained later, is that only large competitors are likely to be selling more than 10,000 copies per year.

### D-Flat Gets a License

I wanted to learn more, so I set out to get a royalty-free license to distribute more than 10,000 copies of D-Flat. I called Borland and asked for their OEM licensing department, which is what the no-nonsense license statement says I should do. The operator connected me with Karen Rogers. When I asked if this was the OEM no-nonsense licensing department, she hesitated, laughed, and asked what my call was about. Karen is in Corporate Affairs. I told her what I needed, and she transferred me to John Smart, Borland's patent lawyer. I told him what I wanted, and he said no problem. When he got my name, company, and the name of D-Flat, he recognized it, knew I was from the press, and we had a congenial conversation about the situation.

Getting the license was easy. I have it now and may distribute D-Flat without restriction. But the disturbing part is to get this license, users, potential Borland competitors or not, had to tell Borland about the product. Open the books, so to speak.

### The Shift Hits the Kahn

Programmers around the world read the 10,000-copy restriction and went ballistic. There are many venues for software distribution where the developer cannot account for numbers. One is shareware. Another is the distribution of royalty-free redistributables that you develop for other programmers to use to develop programs which they distribute. Such as D-Flat. Get's hairy, doesn't it? But for whatever reason, no one gives up freedoms without a fight, particularly when they are taken away in the small print. Programmers felt betrayed and said so, loudly and with some emotion. The Borland forums on CompuServe burned with their complaints. Many vowed to return the package for a refund. Most demanded an explanation.

### Borland Responded

Borland reacted to the outcry by posting a Q&A dialog on CompuServe that was supposed to clear up the matter. They announced their intention to revise the no-nonsense license statement to remove some of the restrictions, but the wording of the Q&A was vaguer than the original no-nonsense license statement, and it was not clear how they would deal with the problem short of removing all of the restrictions.

The 10,000-copy restriction was silly at best. Borland's VROOMM patent is the only one it has, although others are pending. One wonders who Borland was trying to contain. The Q&A document stated publicly that the restrictions

are directed only at certain large, litigious competitors and that others had nothing to worry about. It said, "If you are not a litigious competitor, then the restriction doesn't apply to you." How does Borland know who is going to sue them? It seems to be saying, "If you sue me, I'm taking my license back."

Smart narrowed the number of litigious competitors to two and would not name them but said that one of them was suing Borland now. That would be Lotus.

Listen up, Philippe. Overlays ain't that hard to figure out. Lotus can hire some fast and loose programmers and do their own overlay manager quicker than you can snap-roll your Waco. It isn't worth all this bad public relations just to force them to do that.

Speculation follows. Could be someone at Borland heard that Lotus wrote everything in Turbo C and is heavily committed to some compiler implementation-dependent stuff. That would be the *coup de grace*. Rig your no-nonsense license statement so that a big competitor, one who just happens to be suing your eyes out, cannot upgrade to the next version of their principal development tool. This is the only scenario that I can come up with that even remotely explains Borland's changes in attitude about patents. Nonetheless, I wonder about the ethics and legality of a license that is publicly waived for everyone except certain competitors. Borland told us that there are only two targets, and it gave us enough information to guess who the large, litigious competitors are. End of speculation.

### Other Restrictions: What You Can Compile

The 10,000-copy limit and the patent threat are only the first half of the story. Most programmers did not notice that Borland C++ 3.1's no-nonsense license statement contains language that restricts what kind of programs you can compile and distribute. You are restricted from developing:

> ...a compiler, development tool, environment product or library which includes any of the libraries, DLLs or source code included in this package...[or]...a product that is generally competitive with or a substitute for any Borland Language product.

How many of you 3.1 users knew that? You didn't read your no-nonsense license statement, did you? See what it says? You can't develop a programmer's editor because it would compete with Brief. You can't develop a compiler, an IDE, a profiler, a user-interface class library (such as D-Flat++), a resource compiler, and so on.

The patent stuff, which caught everyone's eye, drew attention to these other restrictions. Most of the programmers spoke out as if the noncompete conditions were new to version 4.0. They were not. Nonetheless, users were mad about the noncompete stuff too.

### The Paradox Paradox

What was the intent? According to Smart, Borland did not want to restrict you in any of the ways that I just described, even though the language in the new no-nonsense license statement said otherwise. It merely wanted to prevent anyone from buying the Paradox engine, putting a user-interface shell around it, and selling a product that competes with Paradox. There's the real paradox, folks. The languages department wants to provide developers with a comprehensive database engine, but the applications department does not want those developers to use it in ways that the company does not approve of.

> *Every programmer understands that "software patent" is an oxymoron*

### More Nonsense

Borland's shot in the foot was a double-barreled blast. The second barrel on their no-nonsense license statement contained the condition that the program you develop "...may not be an operating system."

Wow. I didn't know that Borland was planning to release an operating system. My earlier speculation would not apply to this one. The other large litigious competitor doesn't use Borland's compiler to compile their operating system. That can't be why Borland put this restriction in. My usually reliable sources weren't telling, either, other than to say that one of the lawyers added the language. Makes you wonder. Doesn't anybody outside of the legal department read this stuff before it goes in the big blue and white box?

This operating-system restriction had wide-ranging implications. For one thing, it ruled out UNIX ports. But worse, it hit embedded-system developers squarely between the eyes. An embedded system does not usually use MS-DOS, DR-DOS, or any other general-purpose operating system. The embedded program will be self-contained, which means that it includes an operating system. You couldn't write one of them according to the new terms.

Of course, another outcry was heard 'round the world. Borland reacted quickly by saying in its CompuServe Q&A, "The restriction against creation of an OS is deemed unnecessary and will be dropped."

Why was the restriction necessary one day and not the next? One theory involves Borland's agreement with Microsoft. Borland has a license to distribute certain Windows development materials that are covered by Microsoft's copyright of the Windows API. Without those materials, Borland's users would need to purchase the SDK to develop Windows programs. The theory speculates that Microsoft granted that license on the condition that Borland would somehow prohibit its users from developing operating systems that compete with Microsoft. I do not believe this theory. Microsoft does not have similar restrictions on your use of its own software development products. I believe that Microsoft grants those licenses because its best interests are served when you develop Windows programs regardless of the compiler that you use. No, the story that Smart told me makes more sense. One lawyer, who doesn't know what an operating system is, put the language in, and no one else was smart enough to cross it out.

So, once again, why did the urgency of this operating-system condition disappear so fast? Because it isn't important, and you, their customers, howled, that's why.

### The Healing of the Wound

Borland lost a large measure of credibility during this episode. It stuck a toe in to test the patent waters and got it shot off. It tried for some reason to limit the development of operating systems and got the door slammed shut in its face.

To regain some lost esteem, Borland went into damage-control crisis mode. The spin doctors rewrote the no-nonsense license statement to remove the operating system and 10,000-copy restrictions and to water down the noncompete clause to reflect their true, original, noble intentions, which are more palatable. The only restriction is as follows:

> Your programs may not be merely a set or subset of any of the libraries, code, Redistributables or other files included in this package.

That restriction seems reasonable and reflects Borland's responsiveness to the concerns of its customers. More impor-

tantly, it demonstrates the power of the user's voice when a vendor tries to impose unreasonable restrictions on its customers. We, the programmers, won this one by the sheer force of our numbers. I hope that all vendors are watching and that they will have the good sense to let some users look at what the lawyers write before they commit to it.

We hope that Borland learned that lesson. In its zeal to counter their enemies, it forgot who its friends were. It showed us a different face, one that we had not seen before, a mean-spirited one that holds and can enforce software patents if it wants to. We want to believe that the old face has returned and that it is the true one.

Borland is not a litigious company. It has never sued anyone. Borland thought that reputation would hold it in good stead in the face of public reaction to their actions. But when I asked about the future, when the empty suits change occupants, when I asked about how we could be sure that some future regime would continue to overlook those fascist and burdensome no-nonsense license restrictions, Borland could not answer. In the face of overwhelming public disapproval of their actions and the hidden agenda that those actions seemed to reveal, Borland did what it had to do. It took it all back. The version 4.0 no-nonsense license statement is, if anything, more liberal and more absent of nonsense than that of version 3.1.

### What We Learned

This episode teaches us something else, too. Read the licensing conditions on whatever software-development tool you use to develop a program that you plan to distribute. Virtually all C++ compiler products have some restrictions. They require that you put a valid copyright notice on your software and do not remove any copyright notices that they include on the redistributable components. You indemnify the vendor from any liability if your programs do not work. You may redistribute the redistributables only as a part of an operating program and not as redistributables themselves. You must be a registered user of their product to distribute programs compiled with their product.

Only Borland and Microsoft have restrictions about what the programs themselves may do. Borland does not want you to distribute sets and subsets of its redistributables, whatever that means. Microsoft does not want you distributing programs that use its libraries, MFC, and VBX redistributables in programs that programmers use to build programs that use VBXs. Interestingly, although Symantec C++ Professional licenses the MFC libraries from Microsoft for just such a purpose, it does not have a similar license restriction about what you can do with them. As you can see, it gets complicated.

### Patent-Leather Agenda

Several years ago, every issue of every automobile magazine was sure to have at least one editorial where the author whined about the national 55-mph speed limit. Until the law was repealed those magazines acted as the self-appointed guardians of our right to drive fast. Similarly, every gun magazine today can be depended upon to wedge their editorial agenda against enemies of the people such as Janet Reno and James Brady who would abridge our Second Amendment right to own and bear semi-automatic assault weapons and handguns (in a well-formed militia, of course).

We in the programming-trade press are coming to sound very much like those other self-interest watchdog publications. We are beating this issue of software patents to death. What is the point?

Well, in the first place, the issue is a technical one that is being administered with nontechnical criteria, and we, the trade press, are the only public forum that has or will tell the truth. The arguments for software patents are based in power, money, and politics. When you apply nontechnical, interim solutions to a technical problem, you almost always arrive at a final solution that does not work, if only because the technical parts of the problem are unsolved. Unfortunately, we are singing to the choir. You, our readers, already understand.

Every programmer understands that "software patent" is an oxymoron. The lawyers who prepare and file the patent documents do not understand. Neither do the Patent Office bureaucrats who grant the patents. Some of the people who hold the patents understand, but they are motivated by things other than technical purity, such as the promise of gain.

One of our smartest programmers is Bill Gates. His plans for Microsoft include 100 new software patents per year. He knows better, knows that the system does not know better, and plans to use that advantage to expand his power, influence, and wealth. Why am I surprised? Isn't this supposed to be the greed-is-good generation?

Software patents are everywhere. Most of the software-tool vendors who bring their demos to the *DDJ* conference room proudly announce that they have filed patents on parts of their products. I don't think they read our editorials. I recently attended a briefing of a new version of a well-known database management system. The vendor has a patent pending on his particular use of the B-tree algorithm in the indexes that support interfile relationships. I had to laugh, because years ago I used an identical technique in government software systems. It was obvious then. It is obvious now. The patent will probably be granted.

And yet, we keep thumping the drum. If we educate you about the dangers of software patents and their potential to compromise your livelihood, then we have done some good. At least you will be prepared. If enough of us kick up enough of a fuss, maybe our legislators will get the hint and do something positive about the problem. Maybe we can get the attention of those who need to understand their business and ours a little better. The effort might be in vain, however. Even if we educate the lawyers, they will pretend to continue to operate in a cyberfog. Technical ignorance supports their agenda, which is collecting fees for knowing the law. Educating Patent Office bureaucrats is probably a waste of time, too. As soon as one of them understood software well enough to do the job, they would quit and find work as a programmer. Who wouldn't? And finally, trying to educate wannabe wealthy patent holders and fee collectors is guaranteed to be folly. They have already learned all that they need to know.

### Goodbye, Sonny

I want to tell you about an unsung hero in our industry, someone who will never be the subject of a book, who will never receive a prestigious award, and about whom you will never hear, except today in this column.

Almost six years ago, in my first "C Programming" column, I told how my brother Fred got me started with C. He was a microcomputer pioneer with an engineering degree and a love of programming. He had every issue of *Dr. Dobb's* and was among the first of the homebrew computer makers. He was not one of the famous hackers, but he knew more about it than most. He kept a low profile and kept in touch with everything that was happening. You did not know about him, but he knew about all of you.

It was 1971. I was pounding out Cobol accounting programs when Fred dropped by. He brought a small aluminum hobby box with a front panel sporting four LEDs, four toggle switches, and some push buttons. It was a home-built computer, about the size of a cigar box, running an Intel 4004 microprocessor. I had never seen such a thing. We spent all afternoon loading programs and data into the small memory with switches and buttons and reading the output as binary values in the lights. The 4004 was meant to be used in calculators, but Fred was using it for some kind of black-box application in what we would call today an "embedded system." We got excited about that little box with four data lines and 256 bytes of memory. Someday, we thought, everyone would want one.

Fred grew with and ahead of the technology, always among the first to try new things. He built one of the first Altairs. It's still in his basement lab, still running. He recruited me to write programs for his projects and showed me how to squeeze code into tight spaces, citing Stevens's first law of programming, which said that any program can be reduced by one byte, and Stevens's second law, which said that sometimes Stevens's first law had to be applied recursively. Together we built many diverse embedded systems: a telephone call accounting system, a point-of-sale monitoring device, a power-company remote-station monitoring system, a laboratory etching device. We integrated microprocessors with PBXs, VCRs, TV cameras, cash registers, voice synthesizers, pagers, stepper motors, plating chambers, motion detectors. Fred designed and built the hardware, and I wrote the programs. We worked side by side, days and nights, and every project was a learning experience. Those old 8080 machines served as prototypes for the products and primitive development systems for the firmware. We typed the source code into memory with a Tele-Type terminal, programmed EPROMS from paper tape, and erased them under UV light. We wire-wrapped and soldered and patched and programmed and hand-assembled our way through dozens of one-of-a-kind machines, each one a wonder to behold and every one finished and performing its mission, some of them still in service today.

Twelve years ago, Fred's diabetes took him out of the action. With the passage of time he lost most of his vision, his kidneys, his legs, and a hand to the ravages of the disease. Not able to see well enough to design and debug hardware again, he returned to software, learning UNIX, Forth, C, and assembly language. His reading and typing were slowed sometimes to a crawl, but he never gave up, always maintained a hearty sense of humor, and never lost his enthusiasm for the work. Even when he could barely lift himself out of bed, he talked about ideas for the next project and held onto the belief that he'd lick the odds and see it through one more time. With his right hand gone and unable to see, he was still at it, figuring out how to integrate a joystick keyboard-emulator program with a voice synthesizer so that he could get back to programming as soon as he got well.

On his fifty-sixth birthday, eight days before Christmas, his frail body gave way to a last heart attack, and Fred died, and his monumental spirit, intelligence, and courage were gone.

This is a lonely time for me. Everything that I know and all that I have done that is good can be traced in one way or another to things that my big brother Fred gave me. By his teaching, his example, and his encouragement, he was my mentor, my friend, and my biggest fan. But the loss is not mine alone. He left a family that he loved unconditionally and many loyal and devoted friends. We will all miss him.

**DDJ**

To vote for your favorite article, circle inquiry no. 12.

# Searching for a Search Engine

Tom Swan

Selecting the right tool for the job is always important, whether you are a carpenter, mechanic, or programmer. Too often, however, programmers choose algorithms for the wrong reasons—selecting a Quick sort because they believe it's always the fastest (not true) or using a binary search because they heard it always makes the fewest comparisons when finding elements in a sorted array (also not true). Never choose an algorithm because of its popularity. Depending on your application's requirements, a less-well-known method may be faster or more efficient.

On the other hand, it's human nature to be taken in by claims of superiority, as I discovered while searching for a tool of another variety—I'm talking oil-filter wrenches, now, not algorithms. You see, I need to regularly change the oil and filter in the diesel engine on board my home and sailboat, but it took three tries to find a wrench that would properly unscrew the filter can. The first tool I purchased, the most popular design, came with a band of steel attached to a vice grip that dented the filter case with only minimal pressure. The next sported a plastic strap and the written promise that "one size fits all." Imagine the raw holding power of plastic on a greasy canister, and it's not hard to understand why this filter wrench of the future could never work as advertised. (Products like these make me question whether tool manufacturers ever try their own wares. I often wonder the same about software vendors.) Finally, while poking around in a mechanic's tool chest, I found a homemade pipe, fitted for a socket wrench, with a rough leather strap that grabbed the filter the first time. Later, I bought one from the mechanic. This just goes to show that you should never choose tools based on their popularity or advertising claims. It's often the unlikely junk in the bottom of the drawer that works best.

### Fast Failures

The same is true of algorithms. For instance, in dusting off an old program that I use to prepare Pascal listings for publication, I wondered whether a binary search was the best way to look up entries in a sorted list of keywords—the critical code in this application that parses Pascal programs and converts keywords to lower case, optionally delimited for boldfacing in a word processor. I knew that a binary search makes only *log N+1* comparisons, where *N* is the number of words in the array. Finding an entry in a list of 100 keywords, then, requires a maximum of six comparisons, which I wrongly assumed to be the best results I could expect.

To improve the program's speed, I considered using a hash function or a binary tree to search for keywords, but then I realized that, once again, I had been searching for a search engine for the wrong reasons. Most strings in a program listing are not keywords, so my program's speed was more dependent on how fast a word was not found than it was on the speed of a successful search. In other words, I needed a method that *failed* faster than the competition. Once I came to that realization, I found a way to boost my program's run-time speed by 20 percent. The algorithm that I chose, called a *trie search*—after "information reTRIEval"—is no faster on the average than a binary or hash function, but it requires a maximum of *N* comparisons—where *N* is the number of words beginning with the same letter—to determine that a word is not in the table. In practice, most failed searches take only one or two such comparisons—many take none—far better than required by a binary search, which tends to make the maximum number of comparisons for unrecognized words. By selecting the right tool for the job, taking into consideration the fact that most searches could be expected to fail, I increased my program's speed by using a less-popular, but better-suited, search algorithm.

### Trie-Search Algorithm

A classic trie-search algorithm relies on a table arranged as illustrated in Figure 1. The figure shows only a portion of a complete table, indexed in the first column from A to Z. You could also index the table using other character sets—a standard ASCII trie table, for example, might have 127 rows. Each element in the index contains the number of another array that stores the table's words. The table entries might directly store data, or they could contain pointers—the exact format of the table depends on your program's requirements and the programming language you are using. A zero or null entry in a column indicates there are no words beginning with that letter. There are no Pascal keywords beginning with H, Y, or Z, so those entries are set to zero. (I'm using Borland Pascal's keywords here.)

As you can tell from Figure 1, a program can use a trie-search table to quickly determine whether a search argument is not a key word. In fact, no string comparisons at all are required for entries beginning with H, Y, or Z. Only one comparison is needed to find words beginning with B. To achieve the same results using a binary search requires up to six comparisons for negative searches of Borland Pascal's 57 keywords. In other applications with larger tables, you could extend the algorithm to use two or more tables indexed on a word's successive letters.

| | [1] | [2] | [3] | [4] | [5] | [6] |
|---|---|---|---|---|---|---|
| [a] | 2 | and | begin | far | goto | xor |
| [b] | 3 | array | 0 | file | 0 | 0 |
| : | — | asm | — | for | — | — |
| [f] | 4 | 0 | — | function | — | — |
| [g] | 5 | — | — | 0 | — | — |
| [h] | 0 | — | — | — | — | — |
| : | — | — | — | — | — | — |
| [x] | 6 | — | — | — | — | — |
| [y] | 0 | — | — | — | — | — |
| [z] | 0 | — | — | — | — | — |

***Figure 1:*** *Classic trie-search table.*

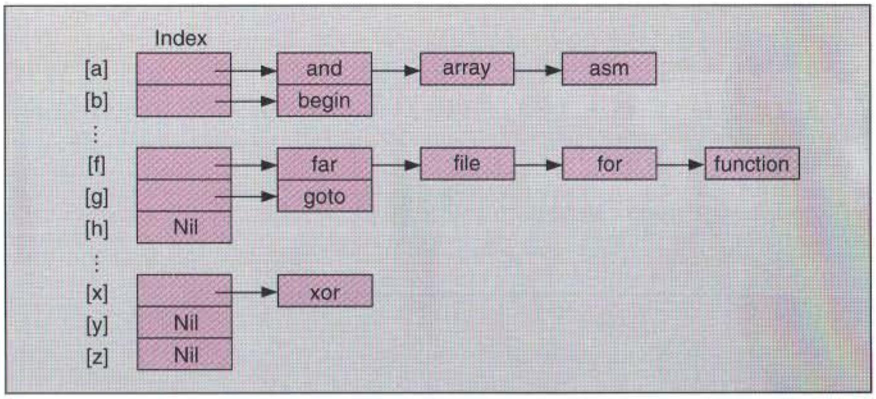



***Figure 2:*** *Trie table converted to a sparse matrix.*

The trick is to minimize the number of full comparisons required to find words in the table or to determine their absence. Once you've structured the table, the rest is easy.

One problem, however, is evident from Figure 1. Many table slots are empty, wasting space. To minimize memory use, you can instead construct the table as a sparse matrix, as illustrated in Figure 2. Now the first column becomes an array of pointers, each of which addresses a list of words beginning with the same letter. (The table could be compressed somewhat by deleting the first letter of each word.) Entries with no words are null. As in the classic table, you could extend the sparse matrix by building other indexes for subsequent letters in each word. Carrying that idea to the extreme reduces the trie table to a digital list—that is, a binary tree of letters, with paths forming the table's words. Small tables such as the one shown here, however, work just as well with a single-level index.

Example 1 is pseudocode for Algorithm #18, Trie Search. The algorithm simply looks up an input argument's first letter in the index, then searches the linked list for a match. Only a single exact-match string comparison is needed inside the inner loop—the key ingredient of this method's speed. A binary search requires alphabetical less-than or greater-than comparisons, further slowing searches for arguments not found.

```
input
  Arg: String;
var
  P: Pointer;
begin
  P ← Index[Arg[1]];
  while(P <> nil) do
  begin
    if P^.Word = Arg then
      return True;
    P ← P^.Next;
  end;
  return False;
end;
```

***Example 1:*** *Pseudocode for Algorithm #18 (trie search).*

## Pascal Parser

Listings One, Two, and Three show the source code for my Pascal Parser, IDENT.PAS. SEARCH.PAS, the Pascal unit in Listing One (page 143), implements the trie-search algorithm. Keyword lists are composed of linked records of type *ResWordRec.* The global *Index* array corresponds to the *Index* column in Figure 2. Procedures *AddList* and *AddWord* build the trie-search tables—you can use these procedures to construct a trie-search engine for any list of words, but the words must be inserted in alphabetical order (see function *Initialize).* Function *IsReserved* determines whether a given word, passed as argument *Ident,* is a member of the table.

The other two listings, COMMON.PAS (Listing Two, page 143) and IDENT.PAS (Listing Three, page 143), use the trie-search engine to parse a Pascal listing. The program converts to lower case all keywords in a Pascal source file, and also optionally capitalizes all non-keywords (specify option –c). Use option –b to add <* and *> delimiters to keywords. The word *begin,* for example, is translated to <**begin**>. Use the –b option only on a copy of a source file—after conversion, the file will no longer compile. (You can restore the original text by deleting all instances of <* and *>.) I use WINWORD.MAC (Listing Four, page 145) in Word for Windows to convert delimited words to boldface after inserting a listing into a document. You could probably whip up a similar macro for other word processors.

## Your Turn

Next month, more algorithms. Meanwhile, send your favorite algorithms and tools to me in care of *DDJ—software* tools, that is.

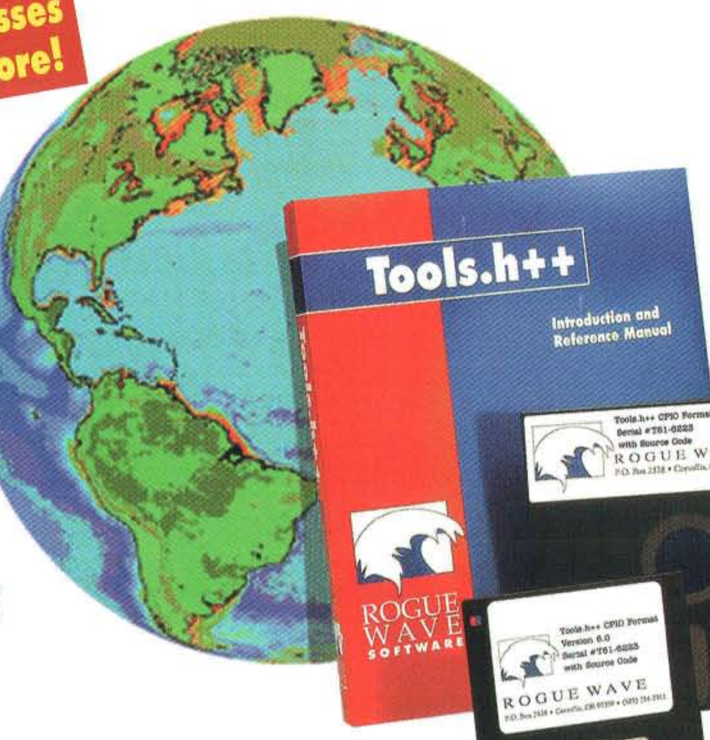
**DDJ**

**(Listings begin on page 143.)**

To vote for your favorite article, circle inquiry no. 13.

# Tools.h++ Version 6.0
# Now Internationalized!

**Collection Classes Plus Much More!**




**Tools.h++**, the best selling industry standard C++ library, now includes Internationalization! A complete, efficient and versatile toolbox of over 100 C++ classes Tools.h++ will make virtually any programming job easier.

**And now new Version 6.0 includes:**

**Internationalization**
- Multi-byte and wide character strings
- Localized string collation
- Parse and format times, dates, and currency in multiple locales
- Support for multiple time zones and daylight savings rules
- Support for localized messages
- Localized I/O streams
- Many locales can be active simultaneously

**Multi Thread safe**
- Safe for use in multi thread environments
- Support for task specific data

**Exception**
- Comprehensive exception hierarchy
- Exception handlers can report in the local language

**Comprehensive test suites are now available!**
- ToolsPro.h++ includes a complete test suite for all classes

***Now you can create one executable and ship it to multiple countries! Tools.h++ lets you read and print times, dates, numbers, and currency in the local format and language!***

***Includes template and non-template classes! You choose!***

**Now also available on WINDOWS NT *and* Macintosh**

**Tools.h++ is a complete, efficient and versatile toolbox of over 100 C++ classes:**

- String and Character manipulation classes.
- Singly and Doubly linked lists, Stacks, Queues and Vectors classes.
- Smalltalk™–like Collection classes: Set, Bag, SortedCollection, Ordered Collection, Dictionary, Stack Queue, etc.
- Regular Expression Class for search and replace.
- Tokenizer Class for easy string parsing.
- File Class to handle file manipulation with read, write, seek, erase, etc.
- B-tree Class to handle efficient keyed access of disk records.
- File Space Manager Class to allocate, deallocate and coalesce space within files.
- Virtual and Buffered Page Heap to manage objects bigger than 64k.
- All objects fully persistent.

Rogue Wave's Tools.h++ is an "industrial strength" library used in many commercial applications, small and large. All classes have not been derived from a single root object, so they can be easily integrated with other class libraries.

Tools.h++ is an excellent example of how to write true C++ code correctly. Source Code is included. All non-Windows classes are strictly portable between DOS and UNIX.

## Other Rogue Wave Libraries Include:

**View.h++**
A complete C++ encapsulation of the industry standard OSF/Motif tool kit.

**LAPACK.h++** and **Math.h++**
High level math libraries.

**DB.h++**
A portable C++ database interface for RDBMS's.

Our libraries work with most of the C++ compilers on the market today.

PURIFY'd

ROGUE WAVE SOFTWARE

# TO ORDER CALL 1-800-487-3217

P.O. Box 2328 • Corvallis, Oregon 97339 • 503-754-3010 • 800-487-3217 • FAX 503-757-6650

CIRCLE NO. 439 ON READER SERVICE CARD

# Think Globally, Act Locally: Inside the Windows Instance Data Manager

Klaus Müller

### Introduction

*by Andrew Schulman*

Many DOS programmers still "don't do Windows," seeing it as irrelevant to DOS programming. But this is unrealistic, because Windows Enhanced mode affects even software loaded *before* Windows is loaded, including DOS memory-resident programs (TSRs), device drivers, and even DOS itself. In short, many DOS programs have no choice but to become Windows-aware.

Windows awareness is especially important for "instance data." For example, load a TSR like Chris Dunford's CED command-line editor and then start Windows Enhanced mode. Open two DOS boxes. Type a command in one DOS box, switch to the other one, and then press the up-arrow key. The command typed in the first DOS box appears in the second, as "state" leaks across! This unintentional interprocess communication might appear to some programmers as a feature, but it is more likely to strike users as a bug. A DOS programmer who blows off this problem with a proud "I don't do Windows" had better be sure that every one of his users feels the same way.

Now put the statement LOCALTSRS=CED in the [NonWindowsApp] section of SYSTEM.INI (if it's not already there), and restart Windows. This time, commands typed in one DOS box are recalled only in that DOS box; they don't leak across into the other one. As its name implies, LOCALTSRS= has somehow made CED's state "local" to each DOS box. Exactly how this works is the subject of this month's "Undocumented Corner."

Rather than use CED, you can switch to the DOSKEY utility that Microsoft includes in DOS 5 and 6. This command-line editor exhibits the same correct behavior as LOCALTSRS=CED, except that no LOCALTSRS=DOSKEY statement is necessary. Clearly, DOSKEY is doing something CED isn't.

Unlike CED, DOSKEY intercepts INT 2Fh and looks for calls to AX=1605h and AX=4B05h. If it receives a call to either of these functions, DOSKEY declares the address and size of its command-line history buffer as "instance data"— that is, as data that must be local (rather than shared) in each DOS box. Note that "instance data" in this context has nothing to do with multiple "instances" of Windows applications (though there are some analogies).

INT 2Fh AX=1605h is documented in the Windows Device Driver Kit (DDK). This is a crucial interface with which DOS programmers must be familiar. It may seem perverse that an API necessary for DOS programmers is located in the Windows DDK, but INT 2Fh AX=4B05h, documented as "Identify Instance Data" in the *MS-DOS Programmer's Reference*, is identical (at least as it relates to instance data). In fact, DOSKEY uses the same piece of code to handle both calls. Unfortunately, the *MS-DOS Programmer's Reference* indicates that INT 2Fh AX=4B05h is related to the relatively unused DOS task switcher and says nothing about the need for DOS programs to instance data for compatibility with the far more prevalent Windows Enhanced mode.

But what does INT 2Fh AX=1605h actually do? And how does it relate to other means of instancing data, such as the LOCALTSRS= statement (or its LOCAL= equivalent for

There are programmers who see Windows not as a graphical user interface, but as an operating system with preemptive multitasking of virtual machines (VMs). These developers have to worry about things like hardware interrupt handlers that operate in the context of a specific VM.

Asynchronous access of data in a specific VM is possible via the documented *CB_High_Linear* field in the largely undocumented VM control block (VMCB) structure (see *DDJ*, January/February 1994). The current VM is mapped in the first megabyte of the linear-address range. You can access any VM's address space (current or not) by adding *CB_High_Linear* to its linear address.

So far, so good. But sometimes when you want to access data in a VM, you get a page fault. This page fault is transparent (applications don't see it), but it can lower performance and lead to serious problems inside an interrupt handler.

What's wrong here? It turns out that the page faults are an integral part of instance-data management in Windows Enhanced mode.

*Klaus studies information technology at the Dresden University of Technology's Fraunhofer Institut for Microelectronic Circuits IMS-2. He is currently developing a heterogeneous multiprocessor system for parallel image processing based on the TMS320C40 processor. Contact Klaus on CompuServe at 100117,2526.*

### Local vs. Global Data

Instanced data is born as global data before Windows starts, but once Windows starts, it becomes local for each VM.

In general, it would be good if *all* data in a VM were local. In a genuine protected multitasking environment, global data is very dangerous. Consider an ugly TSR that crashes a DOS system. In a truly protected environment, the multitasking kernel nukes only the crashed VM.

Windows 3.x Enhanced mode does not support this level of safety. Since Windows is based on DOS, Microsoft compromised. All memory allocated before Windows starts (including DOS TSRs, device drivers, and DOS itself) will be mirrored in each VM via the 386 paging mechanism. Thus, this memory is *global*, shared by all VMs.

There are several reasons to allow the presence of global data. For one thing, Windows rests on top of DOS, and some DOS data *must* be global.

*(continued from page 125)*
DOS device drivers)? Eventually these methods of declaring instance data to Windows, plus several others, lead to the *_AddInstanceItem* function provided by the Windows Virtual Machine Manager (VMM). This call is documented in the Windows DDK and in the book *Writing Windows Virtual Device Drivers,* by David Thielen and Bryan Woodruff (Addison-Wesley, 1994).

Okay, so what does *_AddInstanceItem* do? What is instance data really, and how does VMM implement it? The Microsoft KnowledgeBase includes a surprisingly good explanation, "Instanced Data Management in Enhanced Mode Windows" (Q90796). However, this states that the internal "instance buffers are not accessible to VxDs or TSRs; they are local data structures to be accessed by the VMM only."

In this month's "Undocumented Corner," Klaus Müller shows how to access the internal instance-data structures, using a virtual device driver (VxD) loaded early in the Windows boot process, right after VMM. By using the documented *Hook_Device_Service* call to intercept the *_AddInstanceItem* function, Klaus's VxD builds up a picture of the instance-description buffer.

To further describe the Windows instance-data manager, Klaus uses another interesting method: examining the error-message strings that appear in the widely available debug version of WIN386.EXE. Many of these error messages refer to internal VMM functions whose names we otherwise would not know; see Figure 1.

Once the internal instance-data structures are located, the results must be *interpreted.* Let's say that (as in Figure 2) the virtual keyboard device (VKD) instances 28h bytes at address 415h. So what? Well, these 28h bytes include the BIOS keyboard buffer. VKD instances this buffer so that each DOS box— actually, each virtual machine (VM), including the System VM in which Windows applications run—has its own local BIOS keyboard buffer. Keys typed in a DOS box don't leak into the user's copy of Excel or Word for Windows. This has a downside, too: It's difficult (though not impossible) for Windows applications and full-screen DOS boxes to deliberately "push" keystrokes into each other.

In previous "Undocumented Corner" columns (January and February 1994), Kelly Zytaruk examined the Windows virtual machine control block (VMCB) and noted that offsets 0BCh and 0CCh in the VMCB refer to instance data. Klaus fleshes out this point.

I expected that all of Klaus's results would have to be thrown out for Microsoft's forthcoming Chicago operating system (Windows 4). However, Klaus reports that the instancing mechanism has not fundamentally changed and that his programs for locating the instance-data structures work in Chicago, too. Of course, Chicago is still in prerelease, and anything can happen between now and when it ships. Klaus does note that VMM in Chicago provides a *_GetInstanceInfo* call, which reports whether a given region is instanced or not, though whether this is more useful than the existing *_TestGlobalV86Mem* is unclear. Future articles from Klaus will show how to locate device CB areas and asynchronously access instance data without causing a page fault.

In addition to the usual places to download *DDJ* code (see page 3), you can get programs and source code mentioned in this article from the new Undocumented Corner area in the DDJ Forum on CompuServe (GO DDJ). If you have any comments or suggestions for future articles, please post messages to me there, as well; my CompuServe ID is 76320,302.

*(continued from page 125)*
Consider the example of the DOS system-file tables (SFTs) when SHARE .EXE is loaded. The SFTs present before Windows started need to be visible to all VMs.

Global data also saves memory. Remember the days before 386 memory managers? A well-equipped system with network drivers, mouse drivers, disk-caching programs, and other resident software could consume 400 Kbytes of conventional memory. If you created three VMs, you quickly wasted 1 Mbyte of memory.

Wasted? What about the paging mechanism of 386-mode Windows with its ability to extend physical memory with disk memory? Unfortunately, TSR memory must often reside permanently in *physical* memory. Many TSRs maintain a hardware interrupt; paging out memory which contains hardware interrupt handlers would cause unpredictable results. The interrupt-handler code would be executed for each VM separately, so the best result from paging global TSRs would be a remarkable performance decrease. Consequently, memory belonging to DOS TSRs, device drivers, and DOS itself is, by default, global to all VMs, and is not pageable. If a TSR changes some data in its memory area, the change takes effect in all VMs. If the TSR crashes a VM because of a memory-related error, all VMs will be crashed, including the System VM with all the Windows apps.

Although the executable code of the TSRs can be the same for all VMs, the data must be private for each VM. Such privacy is called "instancing." While this should not be confused with instance data in Windows applications, it is analogous.

Any portion of software loaded before Windows can be instanced using one of several documented techniques, including INT 2Fh AX=1605h and the LOCALTSRS= and LOCAL= statements in SYSTEM.INI. Some obvious candidates for instancing are the interrupt-vector table and parts of the BIOS data area. The keyboard buffer has to be instanced, as do the history buffers belonging to any command-line editors loaded before Windows. (Why doesn't the history buffer for a command-line editor loaded *inside* a Windows DOS box have to be instanced? Answer: Because the memory is *already* local.)

### Instance Data and Paging

The memory management of Windows Enhanced mode is based on the 80386 paging mechanism. The smallest unit of memory is one 4K page. The following types of memory are to be found in the VM's address range (the page types are documented in the DDK):

- Global data. Nonpageable, systemwide data shared by all VMs. Changing the data in one VM changes the data in all VMs. By default, the allocated DOS memory at Windows startup is PG_SYS. The memory is "mapped" into each VM. Page type: PG_SYS.
- Local data. Pageable, local data specific for each VM. The free DOS memory at Windows startup is local. A DOS program started in a DOS box cannot crash other VMs because of an error that belongs to its PG_VM memory. Page type: PG_VM.
- Instance data. A mixture of instanced and global data. Like PG_SYS pages, they are nonpageable. The differ-

ence from global (PG_SYS) memory is that some of the data is marked as local and handled in a specific manner. Page type: PG_INSTANCE.

Though PG_INSTANCE pages are not paged out to disk, PG_INSTANCE can be marked "not-present"; this is key to the instance-data mechanism. Only one VM at one time has a PG_INSTANCE page "present." The corresponding pages in the other VMs are marked not-present. If the pages were all swapped out during a task switch, the global data would become local; the pages would not be updated when another VM changed the global data. Conversely, if the pages were still present in the other VMs, writing data to the instanced part of the pages would make them global because all corresponding PG_INSTANCE pages have the same physical base.

Windows saves the instanced parts of PG_INSTANCE pages in a special buffer. Because the paging mechanism has 4K granularity, a physical copy is required for any instance data item smaller than 4K. An instanced data item can be as small as a single byte. Many individual instance items can thus decrease performance.

## The Instance-Data Manager

The instance-data manager (IDM) manages the instance mechanism. It is part of the Windows Virtual Machine Manager (VMM) and exports the documented *_AddInstanceItem* service. While processing *_InstanceInitComplete*, the IDM allocates memory via the VMM *_PageAllocate* service for the following buffers:

- For each instance item, the instance-description buffer contains its linear address, its length, and the location of its data within the instance buffers. VxDs declare instance data via the documented *_AddInstanceItem* service and *InstDataStruc* structure. The code in VMM for *_AddInstanceItem* builds a linked list of these structures. At *_InstanceInitComplete*, VMM discards any duplicate instance-data requests (for example, if one VxD instances 2 bytes at 415h, and another instances 20h bytes at 400h) and builds the instance-description buffer as a sorted array of *InstMapStrucs*. During *Instance_Init_Complete*, the sorted list is examined. With the information from the list of *InstMapStrucs*, the IDM builds the instance-description buffer as an array of *InstanceMapStrucs*.
- The instance-snap buffer is used to save the instant data present at Windows startup time. When Windows exits back to DOS, the instance data is restored.

*_InstanceInitComplete no instance list*
Tells us there must be an instance list. This is obviously a chain of linked *InstDataStrucs* from documented in VMM.INC.

*_InstanceInitComplete entries not sorted #esi > #edi*
*_AddInstanceItem* sorts the entries and chains them together via the *InstLinkF* and *InstLinkB* fields of the *InstDataStruc.*

*Computed Inst_VM_Buf_Size of 0 _InstanceInitComplete*
*_InstanceInitComplete* must compute a *Inst_VM_Buf_Size.*

*Allocation failure VM1 Inst _InstanceInitComplete*
*Allocation failure Inst snap _InstanceInitComplete*
*Allocation failure Inst Descrip _InstanceInitComplete*
*_InstanceInitComplete* allocates space for the VM1 Inst buffer, the *Inst snap* and *Inst descrip* buffer.

*Fail grow Instance desc buff AllocateInstanceMapStruc*
IDM function *AllocateInstanceMapStruc* tests the size of the instance description buffer and, if needed, grows the size.

*Swap_Instance_Page, 0 in Inst_Page_Owner for page #ecx*
*Inst_Page_Owner* is the VM in which the PG_INSTANCE page is marked present.

*Swap_Instance_PageFS ERROR INSTANCE PAGE > 10Fh*
*ERROR: Re-entered instance copy procedure*
The IDM function *Swap_Instance_PageFS* copies the instance data in the instance buffer of the VM and changes the owner of the PG_INSTANCE page.

*ERROR: Instance fault on suspended VM #EBX*
A suspended VM cannot own a instanced page.

*Instance fault on page with 0 IMT_Inst_Map_Size*
The *Inst_Page_Owner* will change after a page fault on a instance page is detected. The IDM determines the new owner VM and saves the instanced data of the old owner via *Swap_Instance_Page.*

*_AddInstanceItem failed InstDataStruc @#EDI Create_Int2F_Inst_Table*
The internal routine that processes the *InstDataStruc* chain from INT 2Fh AX=1605h is called *Create_Int2F_Inst_Table.*

***Figure 1:*** *Some instance-related error messages from the debug WIN386.EXE.*

```
C:\DDJ\INST>instwalk
V86 API from ListInst.386: Result of _AddInstanceItem Hook.
Summary of allocation calls to the Instance Data Manager.

Name of VxD                             InstLinAddr     InstSize

VKD      : Sys_Critical_Init_Proc       0x00000415      0x00000028
VKD      : Sys_Critical_Init_Proc       0x00000471      0x00000001
VKD      : Sys_Critical_Init_Proc       0x00000480      0x00000004
VKD      : Sys_Critical_Init_Proc       0x00000496      0x0000000B
VMM: _Allocate_Global_V86_Data_Area     0x0002807C      0x00000374
V86MMGR  : Unknown_Service              0x00028435      0x00000006
VTD      : Device_Init_Proc             0x000284C0      0x00000008
VDD      : Device_Init_Proc             0x00000449      0x0000001E
VDD      : Device_Init_Proc             0x00000484      0x00000007
VDD      : Device_Init_Proc             0x000004A8      0x00000004
VDD      : Device_Init_Proc             0x00000410      0x00000002
VCD      : Device_Init_Proc             0x00000400      0x00000008
VCD      : Device_Init_Proc             0x0000047C      0x00000004
DOSMGR   : Device_Init_Proc             0x00000413      0x00000002
DOSMGR   :IGROUP                        0x00000504      0x00000001
DOSMGR   :IGROUP                        0x00000000      0x00000400
DOSMGR   :IGROUP                        0x00001550      0x0000001A
DOSMGR   :IGROUP                        0x0000156A      0x00000772
VMM: _Allocate_Global_V86_Data_Area     0x000286B4      0x00000256
DOSMGR   :IGROUP                        0x00003CE0      0x00000004
DOSMGR   :IGROUP                        0x00001342      0x00000002
DOSMGR   :IGROUP                        0x00004830      0x000008F0
DOSMGR   : Device_Init_Proc             0x00027FD0      0x00000010
DOSMGR: Instance_Device                 0x00001278      0x00000004
DOSMGR: Instance_Device                 0x0001EEC2      0x00000004
DOSMGR: Instance_Device                 0x000283F0      0x00000004
VMM:Create_Int_2F_Inst_Tbl:DOSKEY       0x00017D30      0x00000288
VMM:Create_Int_2F_Inst_Tbl:DOSKEY       0x00018C53      0x00000200
VMM:Create_Int_2F_Inst_Tbl:MOUSE        0x00012158      0x00000889
VMM:Create_Int_2F_Inst_Tbl:MOUSE        0x00012AD7      0x0000010A
VMM:Create_Int_2F_Inst_Tbl:VLM          0x0000DD60      0x0000001C
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x00000500      0x00000002
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x0000050E      0x00000014
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x0000070C      0x00000001
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x00005140      0x00000002
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x000053B0      0x000004C8
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x00001252      0x00000002
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x00001262      0x00000004
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x00001429      0x00000106
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x00001530      0x00000001
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x000021F0      0x00000022
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x000012B9      0x00000001
VMM:Create_Int_2F_Inst_Tbl:SYS DRV      0x000012BC      0x00000002
```

***Figure 2:*** *Sample output from INSTWALK.*

• The VM1 instance buffer initially contains a copy of the data in the instance-snap buffer.

When a new VM is created, it also gets a VM instance buffer, which contains all instanced data for the VM when the PG_INSTANCE pages are set not-present. The offset and handle of the VM instance buffer can be found at offsets 0BCh and 0C4h in the VMCB. In Chicago, the VM instance buffers (including the VM1 buffer) are part of the VMCB and are allocated via the *_Allocate_Device_CB_Area* service. (In a future article, I'll describe a VxD that hooks this call to help enumerate these CB areas.)

*_InstanceInitComplete* is an internal VMM function. While working with the debug version of WIN386.EXE, I found some very informative error messages that include references to this internal function; see Figure 1. A great deal can be learned about the internals of the IDM simply by examining these error messages and pondering what the IDM must require for normal, error-free operation. To verify the interpretation of the error messages in Figure 1, it is useful to inspect the code of the IDM. Since the error messages are issued via *Out_Debug_String* with the string offset in ESI, it is easy to locate the desired code that would issue this message if something went wrong. From there, it is easy to work backwards to the code's normal operation.

To illustrate the handling of the PG_INSTANCE pages, assume that they are currently owned by the System VM (SYSVM, or VM1). When a second VM is created, the VMM begins to schedule the VMs according to their priority. If the VMM schedules from SYSVM to VM2, the PG_INSTANCE pages in VM1 are still present, and the corresponding pages in VM2 are not. If any code in VM2 accesses the PG_INSTANCE pages (via the linear address), a page fault occurs. The IDM copies the SYSVM's instanced data to SYSVM's instance buffer and sets the faulting PG_INSTANCE page of the SYSVM as not present. Then the IDM sets the corresponding PG_INSTANCE of VM 2 present and copies the instanced data from the instance buffer of VM2 to the page.

*There are several reasons to allow the presence of global data*

This instancing mechanism is not complicated. But Microsoft has added an important twist for performance reasons: The physical copy of the instance data to the instance buffers occurs only if the pages are written (but not read) by a VM other than the one which owns the PG_INSTANCE pages. Because they are set not-present for the VM, a page fault occurs and the copy starts. Now we can understand why Microsoft has stated that "fragmented and large instance areas decrease the performance of the swapping mechanism."

In summary, for *write* access VMM will: 1. Copy instance data from the PG_INSTANCE page physical base of the former owner to the VM's instance buffer; and 2. copy instance data from the instance buffer of the owner to the PG_INSTANCE page physical base. For *read* access VMM will copy instance data from the instance buffer of the new owner to the PG_INSTANCE page physical base.

### Spying on Instance Data

For a deeper understanding of the instancing, it is useful to write a program that displays the address and size of all instance data and the name of the program which asked for the data to be instanced. LISTINST.386 (see Listing One, page 146) is a VxD I wrote that initializes shortly after the VMM, before other VxDs gain control, and hooks the VMM *_AddInstanceItem* service at *Sys_Criti-*

*cal_Init* time. INSTWALK is a DOS program that displays the information saved by LISTINST.386.

Two other required VxDs are VXDQUERY.386 and (to examine instance data in Chicago) LISTCALL.386. All three VxDs are loaded automatically if you run the DOS program VXDLOAD.EXE just before starting Windows. (VXDLOAD uses the versatile INT 2Fh AX=1605h interface to tell Windows to load the VxDs.) While there isn't room to show all of the source code, the programs are available electronically (see "Availability," page 3).

For each call to *_AddInstanceItem*, LISTINST stores the address of the caller and the offset of the passed-in *InstDataStruc*. (This structure is documented in VMM.INC and INT2FAPI.INC, included with the DDK and with VxD-Lite.) LISTINST has a V86 API which allows DOS programs running in a VM to get the results of the *_AddInstanceItem* hook. INSTWALK uses this V86 API and prints out all instanced data.

Sample output is shown in Figure 2. To clarify what this output means, Figure 3 shows a hex dump (using the PROTDUMP utility discussed in the "Undocumented Corner," January and February 1994) of one of the instance data items declared by DOSKEY; clearly, this instance data is the DOSKEY-command history buffer. By specifying a VM number on the command line, PROTDUMP can view this buffer in each VM; see #1 and #2 in Figure 3. The ownership and purpose of the buffer is identical for all VMs, but the data (here, the command history) differs in each VM. This is precisely what instance data means.

The INSTWALK utility has a –p switch to dump out the instance page-ownership array (an internal IDM structure which lists all PG_INSTANCE pages and their current owners) and the instance-description buffer in raw form. Because this structure is not accessible, I built my own array. Remember, only one VM at a time can own a PG_INSTANCE page. So you have only to walk down the VM's page tables to determine the VM in which the PG_INSTANCE page is set present; see LISTINST.386 for details. This is similar to output from the .mi command available in debuggers such as WDEB386 and Soft-ICE/Windows when the debug WIN386.EXE is installed. INSTWALK –p also dumps out the offsets in the instance buffers, which is important if you want to access the data in the instance buffers.

LISTINST.386 gets the names of the callers to *_AddInstanceItem* via a service provided by VXDQUERY.386, which provides a complete map of the VMM and VxD address space. VXDQUERY

```
C:\DDJ\INST>instwalk | grep DOSKEY
VMM:Create_Int_2F_Inst_Tbl:DOSKEY   0x00017D30      0x00000288
VMM:Create_Int_2F_Inst_Tbl:DOSKEY   0x00018C53      0x00000200

C:\DDJ\INST>\ddj\protdump\protdump #2 18c53 200
00018C53 | 45 00 63 3A 5C 65 70 73 5C 65 70 73 69 6C 6F 6E | E.c:\eps\epsilon
00018C63 | 2E 65 78 65 20 24 2A 00 69 6E 73 74 77 61 6C 6B | .exe $*.instwalk
00018C73 | 20 3E 20 69 6E 73 74 77 61 6C 6B 2E 6C 6F 67 00 |  > instwalk.log.
00018C83 | 67 72 65 70 20 44 4F 53 4B 45 59 20 69 6E 73 74 | grep DOSKEY inst
00018C93 | 77 61 6C 6B 2E 6C 6F 67 00 5C 64 64 6A 5C 70 72 | walk.log.\ddj\pr
00018CA3 | 6F 74 64 75 6D 70 5C 70 72 6F 74 64 75 6D 70 20 | otdump\protdump
00018CB3 | 31 38 63 35 33 20 32 30 30 00 5C 64 64 6A 5C 70 | 18c53 200.\ddj\p
 ... etc. ...

C:\DDJ\INST>\ddj\protdump\protdump #1 18c53 200
83018C53 | 45 00 63 3A 5C 65 70 73 5C 65 70 73 69 6C 6F 6E | E.c:\eps\epsilon
83018C63 | 2E 65 78 65 20 24 2A 00 63 64 20 74 61 70 63 69 | .exe $*.cd tapci
83018C73 | 73 00 74 61 70 63 69 73 00 63 64 5C 70 72 6F 63 | s.tapcis.cd\proc
83018C83 | 6F 6D 6D 00 70 63 70 6C 75 73 00 63 64 5C 74 61 | omm.pcplus.cd\ta
83018C93 | 70 63 69 73 00 74 61 70 63 69 73 00 67 72 65 70 | pcis.tapcis.grep
83018CA3 | 20 4E 46 5F 20 5C 62 6F 72 6C 61 6E 64 63 5C 69 |  NF_ \borlandc\i
83018CB3 | 6E 63 6C 75 64 65 5C 74 6F 6F 6C 68 65 6C 70 2E | nclude\toolhelp.
83018CC3 | 68 00 65 78 69 74 00 63 64 5C 69 6E 73 74 00 63 | h.exit.cd\inst.c
 ... etc. ...
```

***Figure 3:*** *Examining DOSKEY's instance data.*

can name all known VxD services by name and address, start and end of VxD objects and, particularly important for this article, all VxD Control procedures. Even debuggers such as WDEB386 don't provide as complete a map as VXDQUERY. Since VXDQUERY is a commercial program of mine, source code is not provided; however, I'm making a special version available electronically for *DDJ* readers; see page 3.

To use VXDQUERY, another VxD simply loads an address into the EDI register and calls *VxdQuery_Address_To_VxD_Name*. The service returns the name of the VxD plus the closest known procedure that precedes the specified address. If you want to spy on the usage of a VxD call, as I did with *_AddInstanceItem*, write a VxD that uses the documented VMM function *Hook_Device_Service* to intercept the service at *Sys_Critical_Init* time, and collect the address of the callers plus any related parameters. You may need to initialize right after VXDQUERY. During *Init_Complete* or later, you can call VXDQUERY to translate your addresses into VxD names. Finally, your VxD should export a V86 or PM API to make your results public to the non-32-bit world (that is, to a program similar to INSTWALK).

Figure 2 (from LISTWALK) shows all instanced data with their linear addresses. At *Sys_Critical_Init* time, for example, VKD instances 28h bytes at 415h, one byte at 471h, 4 bytes at 480h, and 0Bh bytes at 496h. To make any sense of these numbers, you need to consult a reference that describes standard PC absolute-memory locations. A good source is the file MEMORY.LST included with Ralf Brown's "Interrupt List" (see IBMPRO library 5 on CompuServe); another source is *Undocumented PC* by Frank van Gilluwe (Addison-Wesley, 1994). In the VKD example, the 28h bytes at 415h include the keyboard buffer and head and tail pointer, 471h is the Ctrl-Break flag, 480h points to the keyboard buffer, and 496h includes keyboard-status bytes. It makes sense that VKD wants to instance these portions of the BIOS data area.

Of particular interest are the final calls to *_AddInstanceItem* in Figure 2. VMM made them from an internal routine called *Create_Int_2F_Inst_Table*; Figure 1 explains where this name comes from. This is the VMM code that processes the *Win386_Startup_Info_Struc* chain passed back from INT 2Fh AX=1605h; this documented structure includes an *SIS_Instance_Data_Ptr* field listing items that software loaded before Windows (such as DOSKEY) wants instanced.

VXDLOAD.EXE determines the names of TSRs which supply a Win386_SIS. LISTINST.386 gets a pointer to the list of all SISs on entry in the *Sys_Critical_Init* procedure in EDX, because VXDLOAD fills the *Win386_SIS* for LISTINST.386 with a pointer to its reference data.

VMM first lets all virtual devices instance their data and then calls *Create_Int_2F_Inst_Table* to convert the instance structures provided by DOS TSRs via their *Win386_SIS* to the VMM (IDM) instance structures. The result is a doubly linked list of instance-data structures. You can walk the chain during *Init_Complete* with the *InstLinkF* and *InstLinkB* pointers. After the linked list is complete, the instance-description buffer will be initialized with the data from the linked list. Finally, the so-called "snapshot" is taken in order to save the start-up-time values of the instanced data in the instance snap buffer. If a new VM is created, the IDM creates a VM*x* instance buffer (where *x* is the VM ID) and copies the contents of the instance snapshot buffer into it.

There is one problem with this technique of hooking *_AddInstanceItem* to build up a picture of the instance description buffer. As noted, LISTINST.386 hooks *_AddInstanceItem* as early as possible: at *Sys_Critical_Init* time, right after VMM, and before other VxDs. Unfortunately, this is too late to intercept the *_AddInstanceItem* calls that VMM makes to support the LOCALTSRS= statement. However, LISTINST is still able to find these instance items by following the doubly linked list of *InstDataStrucs*. VMM will instance the entire program, excepting its environment segment.

It would also be useful to determine the ownership of the PG_INSTANCE pages. Again, only one VM at a time can own a PG_INSTANCE page. LISTINST.386 provides the array to LISTWALK. Type LISTWALK –p to dump the instance page-ownership array and the instance-description buffer. The output is more interesting if, immediately after INSTWALK starts, you switch to another session or compile something in the background. In this case, the instance pages are owned by different VMs.

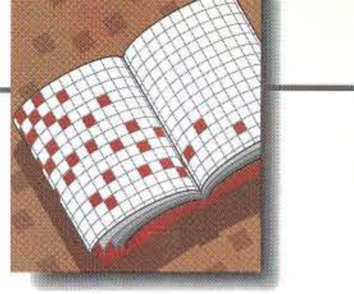
### Not Just Spying

What can you do with this information? In addition to a better understanding of how instance data works and what sorts of data must be instanced, the information presented here might lead to some interesting techniques for accessing instance data without causing page faults. This will be taken up in a future article.

**DDJ**

**(Listing begins on page 146.)**

To vote for your favorite article, circle inquiry no. 14.



# A Clear Look Through Bleary Eyes at Two Books on Algorithms

Tom Ochs

*Algorithm: A set of well-defined rules for the solution of a problem in a finite number of steps.*

Books on algorithms are important tools for the professional software developer. However, as can be seen from the definition, the breadth of issues covered under the heading of algorithms can make the selection of the proper book difficult. Topics can cover numerical applications, business applications, data structures, searching, sorting, optimization, and many others. Some generic characteristics seen in algorithm-related books (ARBs) include: How algorithms are designed, why a particular algorithm is chosen, what measures are used to assess algorithm effectiveness, construction considerations, instances of the algorithms, application examples, test cases, reliability issues, comparisons with other algorithms, and data-structure dependence. ARBs also have an associated level of difficulty that can range from introductory through intermediate and advanced, to specialized-advanced (where you, the author, and three others in the world are interested in the topic). The tone can vary from practical to academic, and the presentation, from well written to just plain poor quality.

Clearly, the selection of a book on algorithms is situational, depending on your needs of the moment. A lot of books are collecting dust on my bookshelves because their characteristics don't meet my current needs. We should take the opportunity to use our analytical skills to determine our needs and then compare those needs to the characteristics of the available books. This can help make our investments in time and money work for us. To supplement your needs assessment, here is my analysis of two relatively new books. Like movie critics who rate films from one to four stars, I will use a π rating, with π being a book that has little to offer, ππ being a book with marginal impact, πππ having significant contribution, and ππππ being a book that *must* reside on the serious developer's shelf.

***Programming Classics: Implementing the World's Best Algorithms***

*Ian Oliver*
Prentice Hall, 1993, 386 pp. $38.00
ISBN 0-13-100413-1

***Algorithms from P to NP, Volume I: Design and Efficiency***

*B.M.E. Moret and H.D. Shapiro*
Benjamin/Cummings Publishing, 1991, 576 pp. $41.95
ISBN 0-8053-8008-6

## Programming Classics

Any book that purports to be "detailing the best algorithms ever devised for a wide range of practical problems..." has a huge challenge ahead of it just to live up to the propaganda on the jacket. Unfortunately, *Programming Classics: Implementing the World's Best Algorithms,* by Ian Oliver, falls far short of the hype. Even though it does cover a wide selection of applications, the coverage is spotty, sometimes shallow, and generally incomplete. In trying to limit the complexity of the presentation, Oliver has also limited its usefulness. On numerous occasions he resorts to hand waving such as: "...is beyond the scope of this book...," "We will not analyze in detail...," "Do not use this algorithm unless you know what you are doing," and "Given the mathematical sophistication needed for dealing with eigenvalues, no discussion of the reasons why the algorithm works will be given." Oliver has mistakenly tried to keep the presentation at an introductory level while introducing intermediate-level algorithms and concepts.

The lack of detailed discussion on the theory of operation of many of these algorithms leaves you to accept Oliver's choice for the implementation based on faith alone. If you have to modify, debug, or optimize the functions, the presentation in this book is generally inadequate. The inconsistency in the amount of detail is illustrated by the adequate coverage of sorting methods, including performance comparisons and application-specific suggestions, while the section on arithmetic is devoid of explanation.

In the poorly explained section on arithmetic, rational methods are introduced and a warning is given:

> "...the methods will fail when integer overflow occurs. For certain practical applications it will be necessary to implement the algorithms in multiple precision arithmetic. The algorithms for multiple precision calculations are beyond the scope of this book."

What Oliver doesn't say is that the methods *generally* fail after only a few operations due to overflow, and the use of greatest-common-denominator (GCD) reduction is only temporarily effective at preventing the overflow. His presentation also skirts the fact that this implementation only works for toy problems if multiple-precision arithmetic is not used.

The author uses his own generic language, reminiscent of Ada, to define his "code" examples. His intent was to produce a broadly targeted representation that was language independent. Instead, it will be difficult to translate some of the code to older languages such as Fortran, Cobol, or C. The example code exhibits problems with initialization, typing, character/byte access, parameter passing, memory usage, and other implementation issues. Since these issues

*Tom is a consultant specializing in the integration of modern software-development methods into technical organizations. He has over 15 years experience as a research scientist, has written a commercial numerical package, and is a registered mechanical engineer living in Albany, Oregon. Tom can be contacted on CompuServe at 70511,652.*

are addressed in a generic way, it is almost assured that few real languages will come close to mapping transparently to his representations, forcing the users to modify their implementations without a clear understanding of the algorithm-design issues. If Oliver was serious about producing reliable code for readers to use directly, he should have chosen specific target languages so the syntax questions could have been dealt with in his implementations.

I rate *Programming Classics* π, for poor execution of a fundamentally good concept, useful only as the first place to look to find references to more complete explanations of the problems to be solved. While this book could be useful to experienced designers looking for a reference that gives terse overviews and points to other sources for details, it will be a hazard for the inexperienced designer looking for a quick method to solve a poorly understood problem.

### Algorithms from P to NP

*Algorithms from P to NP* is a careful, academic text designed for graduate students, upper-level undergraduate students, and computing professionals prepared to use rigorous mathematical analysis in problem solving. If you aren't comfortable with set notation, discrete mathematics, data structures, calculus, and algebraic expression of problems—pick another book. *Algorithms from P to NP, Volume I, Design and Efficiency,* by Moret and Shapiro, is clearly designed as an advanced textbook to be used in a classroom setting with an instructor and does an excellent job in that context. It also serves as a good refresher and reference for those who have been through similar advanced courses. I particularly liked the presentation and felt that Moret and Shapiro did a good job of leading the student through the solution process; however, it is industrial-strength analysis and not for the faint-of-heart. But it is worth the effort. The authors expect you to recognize standard algebraic notation, but introduce specific concepts with which you might not be familiar—a spanning tree, O-notation, generating functions, and directed graphs. The exercises range from simple examples to thesis-level assignments.

*Algorithms from P to NP* concentrates on combinatorial optimization problems and takes a thorough, depth-over-breadth approach. Moret and Shapiro start with several traditional problems such as the knapsack problem (filling a knapsack with the optimal mix of things for a camping trip) and the traveling salesperson problem (traveling through a series of cities while optimizing time or distance). These problems are revisited throughout the book in generic instances as the problem-solving approach is modified and expanded to encompass extensions of the problems. You're given more tools to deal with increasingly difficult examples of the problems as the book progresses.

The reference to a "stack of punch cards," which most graduate students have never seen, dates the origin of some of the examples while demonstrating the timelessness of the problems. Throughout the book, there is just enough nerd humor (my favorite kind) to liven up a graduate course in algorithms. The basic approach of the book is one that I am comfortable with: "The study of algorithms cannot be dissociated from the study of problems." Their approach is to start with problem solving and then show how the solutions map naturally into an algorithm for the effective solution of the problems. They spend time reviewing methods for assessing algorithm run time, but they deal only peripherally with the concept of reliability. This limited discussion of reliability is probably related to the focus on combinatorial problems, as opposed to numerical issues. *Algorithms from P to NP* discusses not only the theoretical, asymptotic behavior of the algorithms, but also the application and implementation issues that impact performance. The language used for example code is Pascal, and the code examples have been used and tested in classroom situations.

The name of the book reflects the concentration in this volume on problems that have solutions which, as a worst case, require "Polynomial time" ($O(N^k)$, where $N$ is the number of items and $k$ is some constant) for their completion. These are represented as P-problems. The second volume deals with NP-complete problems (problems for which no solution has been found that can be completed in polynomial time). NP-complete problems are an ongoing subject of research, and are generally solved by approximation methods.

I rate this book πππ, for concise, clear explanations of problem-solving issues. A serious textbook for serious study of combinatorial issues. Don't pick this book up for light reading!

(For reviews of 14 ARBs dealing with numerical issues, refer to my "Building Blocks" column in the former *Computer Language* magazine, November, 1992.)

**DDJ**

To vote for your favorite article, circle inquiry no. 15.

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

*The advertiser prefers to be contacted by phone; consult ad.
**See individual ads for RS#.

DR. DOBB'S JOURNAL

# PRODUCT DEBUT

*Call Cheryl Canion today at (415) 358-9500 to develop a customized advertising campaign tailored to your marketing needs. Turn the page for more products.*


*Available on our BBS!*
*Another excellent game written with*

## Fastgraph™

Devasoft has released an excellent preschooler's edutainment program:
***Amy's Fun-2-3 Adventure!***

Sure to be a hit among the younger set! Written with Fastgraph, of course.
***Fastgraph Programmer's Graphic Library $199***

Unlock the secrets of graphics programming with the graphics library preferred by game programmers!

**C • C++ • Pascal • Basic • Fortran**

**Ted Gruber Software**
P.O. Box 13408
Las Vegas, NV 89112

Voice (702) 735-1980
FAX (702) 735-4603
BBS (702) 796-7134

**Visa/MC/COD** **Demos available** **No royalties**

CIRCLE NO. 1211 ON READER SERVICE CARD

## Time Arts *Signature*

Graphics Class Library

**The object-oriented framework for all your future graphics development.**

Time Arts

800-959-0509
707-576-7722
Fax: 576-7731

1425 Corporate Center Parkway
Santa Rosa, CA 95407-5434

CIRCLE NO. 1213 ON READER SERVICE CARD

## Offer the HyperHelp Advantage


## and be in good company!

WordPerfect • AT&T
Computer Associates (20/20) • Sybase
Siemens • EDS • Swiss Bank
Merrill Lynch • Corel (CorelDRAW!)
SPSS • Lotus (Ami Pro)

HyperHelp™ - The Industry Standard,
On-line Help Facility for Motif

Bristol Technology Inc.

Ph: (203) 438-6969 Fax: (203) 438-5013 email: info@bristol.com
FTP a demo: bristol.com, demo directory

CIRCLE NO. 1212 ON READER SERVICE CARD

## EOS VBX v1.0

**Visual Basic Programmers can now access all the power of the Evolutionary Object System from a new Genetic Algorithm Custom Control!**

Both Visual Basic and C++ programmers can now create Genetic Algorithm applications faster than ever. This Visual Basic wrapper around EOS for C++ gives a wider spectrum of developers access to a technology that Alan Kay has said, "...is one of the greatest intellectual happenings in computer science."

**EOS** and **EOS VBX** provide all the essential ingredients for building robust GA based applications. These include 3 types of encodings, multiple chromosomes, many different genetic operators, several selection methods, several reproduction strategies, elitism, and much more than we can describe here! In addition both products provide access to Man Machine Interfaces' exclusive adaptive technology where your GA adapts to the problem as it solves it. Please call for more technical details.

**Don't miss out on this limited time discount pricing!**
EOS VBX $139.00 EOS for C++ $139.00
**EOS Professional (EOS C++ & VBX) $199.00**
Also ask about The Financial GENEius Tool Kit
for building GA based Trading Systems!!!

**Order Today: 516-249-4700 • VISA • MC • Check**
**Man Machine Interfaces, Inc.**
**555 Broad Hollow Rd. Suite 230 Melville NY 11747**
**516-249-4700 Fax: 516-420-4085**

***We are The Natural Selection for Genetic Algorithm Applications!***

CIRCLE NO. 1214 ON READER SERVICE CARD

PRODUCT DEBUT

DR. DOBB'S JOURNAL

# MARKETPLACE

## ASSEMBLY

### Free Pentium Optimization Guide

Call or FAX for a copy of Quantasm's Pentium Optimization Guide. Used by itself or with ASMFLOW Professional, you can obtain amazing performance increases. ASMFLOW automatically produces flow charts, call trees register analysis, data x-ref, timing and more. Available for 80x86, 8051, 8085, 8096, Z-80, 6502, 68HC11 and 1750A. ($199.95) Call for free demo and catalog of assembly, TSR and floating point libraries. ($99.95 to $299.95)

**Quantasm Corp.**
19672 Stevens Creek Blvd #307-D
Cupertino, CA 95014
800-765-8086 or 408-244-6826
FAX: 408-244-7268

CIRCLE NO. 1111 ON READER SERVICE CARD

## BUSINESS OPPORTUNITIES

### Earn $4,000 Per Month From Your Home With A Computer!

*FREE CBSI 486 Computer*

Quit spending money on your computer and let it earn money for you. This is a proven turnkey business an individual or couple can run. If you purchase our software and business program, we will give you the computer and printer. If you already own a computer, you may receive a discount. Begin part-time and still retain the security of your present position. We will provide free, home office training. Financing available.

*To receive free cassettes and color literature, call toll-free:*

1-800-343-8014, ext. 1443

*Or Write:*

Computer Business Services, Inc.
CBSI Plaza, Ste. 1443, Sheridan, IN 46069

CIRCLE NO. 1112 ON READER SERVICE CARD

## BOOKS

### *Earn Your High-Tech Degree While Working Full Time*

New book High-Technology Degree Alternatives shows how to get your college degree without quitting your job, attending night school for years, or breaking your budget. All degree programs are accredited, and many do not require classroom attendance. Use VISA or M.C. $21.95 + $3.75 s/h. Professional Publication Inc., Dept. 809.

***(800) 426-1178***

CIRCLE NO. 1113 ON READER SERVICE CARD

## BUG TRACKING

### Bug Tracking Made Easy

**Soffront® TRACK™** is a client-server based ready-to-use Bug-tracking and Technical Support system for Windows™. Using TRACK you can record a Bug, System, and Customer information, query, produce reports, notify other users, keep a log of change history, import, export, and more. Customize forms, fields, views and reports to meet specific needs of your organization.

**Soffront Software**
1806 Milmont Drive, Suite # 169, Milpitas, CA 95035
Call **1-800-SOF-FRONT/(408)263-2703**
or Fax (408) 263-7452

CIRCLE NO. 1114 ON READER SERVICE CARD

## C PROGRAMMING

**Free Report** on how you can Eliminate 87% of **C** Programming Errors in less than 2 hours!

This valuable report reveals secrets known by less than 5% of all professional programmers. These little known but powerful techniques prevent almost 90% of all design and maintenance errors with the guarantee that your compiled code will be virtually error free. Limited quantity available.

**Logic Technologies**
1 (619) 228-9653 FAX 1 (619) 369-1185
56089 29 Palms Hwy, Ste 254-DD, Yucca Valley, CA 92284

CIRCLE NO. 1115 ON READER SERVICE CARD

## CD-ROM

### 640 MB OF SOURCE FOR LANGUAGES & OPERATING SYSTEMS FOR $39.95

The LARGEST COLLECTION of source for compilers, function libraries, and docs for standard and research languages & OS ever assembled.

...Ada, Basic, C, C++, E, ET++, Lisp, Mach, Modula, Oberon, Pascal, Prolog, Rexx, Sather, Scheme, Self, Simula, Tcl, Yacc, etc...

| | |
|---|---|
| Language/OS CD-ROM | $39.95 |
| Other Titles Available | |
| Graphics – 1 (Applications w/ source) | $24.95 |
| Audio – 1 (Applications w/ source) | $24.95 |
| Multimedia – 1 (Applications w/ source) | $49.95 |

**Knowledge Media**
**436 Nunnely Rd., Ste B, Paradise, CA 95969, USA**
**1-800-78-CD-ROM VISA/MC/COD**
**+1-916-872-3826 Voice/FAX**

CIRCLE NO. 1116 ON READER SERVICE CARD

## CD-ROM

### Your Best CDROMs!

| | | |
|---|---|---|
| Cica MS Windows CDROM | 4000 new Windows programs - quarterly updates | $29.95* |
| Giga Games CDROM | 3000 hot Games for MSDOS and Windows | $39.95* |
| Space and Astronomy CDROM | Thousands of NASA images and data files | $39.95* |
| C User Group Library CDROM | A collection of user supported C source code | $49.95* |
| Simtel MSDOS CDROM | Classic: 650 MB Shareware / Freeware for MSDOS | $29.95* |
| QRZ! Ham Radio CDROM | FCC Callsign Database for Win & DOS + files | $29.95* |
| Gifs Galore CDROM | 6000 mostly 640x480 GIF Images in every catagory | $39.95* |
| Gutenberg Project CDROM | Classic Literature and historical documents | $39.95 |
| Hobbes OS/2 CDROM | 600 MB always current Shareware/Freeware for OS/2 | $29.95* |
| Source Code CDROM | 650 MB of Unix & DOS source code for programmers | $39.95* |
| Internet Info CDROM | Thousands of computer, net, and internet documents | $39.95 |
| X11/GNU CDROM | X Windows and GNU software for Unix and SPARC | $39.95 |
| Aminet Amiga CDROM | 650 MB new Shareware/Freeware for Amiga | $29.95* |
| GEMini ATARI CDROM | 650 MB 3000 Programs for Atari - also for BBS use. | $39.95* |
| Ada CDROM | Programming tools, Ada source code and docs | $39.95* |
| Nova for NeXTCDROM | 600 MB black NeXT app's, src., docs, etc | $59.95* |
| Nebula for NeXTSTEP Intel | 600 MB NeXTSTEP Intel app's, docs, etc | $59.95* |
| Garbo MSDOS/Mac CDROM | MSDOS and Macintosh Shareware/Freeware | $29.95* |
| Fractal Frenzy CDROM | 2000 high resolution fractals - royalty free | $39.95 |
| FreeBSD CDROM | Berkeley BSD. 32bit O/S for PC, w/ GNU & X11. Src | $39.95 |
| Linux CDROM | Yggdrasil Linux. 32bit O/S for PC, w/ GNU & X11. Src | $49.95 |
| La Coleccion CDROM | MSDOS, OS/2, Windows. Spanish descriptions | $39.95* |
| Sprite CDROM | Berkeley's experimental Distributed OS for Sun's | $29.95 |
| CDROM Caddies | Standard type, highest quality, Lifetime guarantee! | $ 4.95 |

**1-800-786-9907**

All our CDROMs are unconditionally guaranteed.

*Shareware programs require separate payment to authors if found useful.

**Walnut Creek CDROM**
4041 Pike Lane, Suite D-391 Concord, CA 94520
FAX: 1-510-674-0821 email: orders@cdrom.com

CIRCLE NO. 1117 ON READER SERVICE CARD

## CONVERSION

### Basic to C

BAS_C v6.0 translates BASICA/QBASIC/ Quick BASIC source code to C source code automatically. The translated C source code is scoped, indented, structured, syntax error free and can be compiled Turbo/Borland/Microsoft/Quick C. Runtime C source code is available. 90% conversion ratio. From $599. Call for non-MS Basic.

**Gotoless Coversions**
7105 Dee Cole Dr.
Colony, TX 75056
(214) 625-2323
BBS: (214) 625-6905 FAX: (214) 370-2612

CIRCLE NO. 1118 ON READER SERVICE CARD

## DEVELOPMENT TOOLS

MENAI

**Menai Corporation**

# MARKETPLACE

## NETWORK

# Little Big LAN

Start small, grow as you do

* Peer to peer LAN to 250 nodes
* $75 total software cost/LAN, not node!
* Use Ethernet or Arcnet for speed
* or Zero-slot via serial/parallel/modem
* Share most anything in DOS or Windows
* Needs only 40K of ram

imodes
Information Modes
Drawer F, Denton TX 76202
817-382-7407 FAX
1-800-628-7992 Orders

CIRCLE NO. 1129 ON READER SERVICE CARD

## SECURITY

# CRYPKEY SOFTWARE LICENSING SYSTEM

*"Hardware key like protection without the hardware key"*

CrypKey is a **software protection tool,** offering
- complete security from any disk copy program
- complete compatibility with any MS DOS or MSWINDOWS31 based machine
- complete invisibility—no disk key, no hardware key, less support calls
- instant disaster recovery

CrypKey is a **sales tool,** allowing you to sell your program
- by increments—enable the options the customer purchased
- by number of runs—e.g., sell 100 calculations for $499.00
- by time period—e.g., lease or demo your program for 60 days

CrypKey uses a numeric key that can be transmitted by phone or fax.
Sell your customers more options, more copies, more time or more runs instantly, just by making a telephone call. (Great for overseas customers or distributors.)

*"NOW AVAILABLE FOR NETWORKS"*

CRYPKEY IS PRODUCED BY KENONIC CONTROL—ENGINEERING AND SOFTWARE SINCE 1972.
***Kenonic Controls Limited • 7175 – 12th Street South East • Calgary, Alberta, Canada T2H 2S6 • (403) 258-6200 • fax: (403) 258-6201***

CIRCLE NO. 1134 ON READER SERVICE CARD

## UTILITIES

# THE FILE ANALYST

Dump * Search & Replace * Display
ASCII * EBCDIC * xBase * Delimited
Analyze * Reformat * Translate * Split
Convert * Copy * Scan * Print
Verify * Undo Labels * And more!

20 menu functions to quickly identify, reveal, manipulate and view any data structure — no matter how complex. Perfect testing & import tool for uniform fields and records. Just fill in the blanks & run. No programming! $169.00 Call for Demo.

**212-779-7667**
**Ralph Garner Associates, Inc.**
99 Madison Avenue, New York, NY 10016

CIRCLE NO. 1139 ON READER SERVICE CARD

## OBJECT ORIENTED

***Why Can't Documenting Software be as Easy as Writing It!***

It Can be by Using
***BOCS***

The only CASE Tool giving you the freedom and ease you've grown accustomed to when it comes to coding.

- Inheritance
- Consistency Checking
- Automatic Document Generation
- "Pretty Printing"
- System Level Features
- Traceability, Portable Libraries, 6 Models
- Extensive On line Help

***And Lots More***

***Put an end to documentation drudgery!* Only $595**
Includes both C++ and Smalltalk Code Generation

Call for a demo copy
**Call 301-417-9884 or Fax 301-417-0021**

**Berard Software Engineering, Inc.**
**902 Wind River Ln., Suite 203, Gaithersburg, MD 20878**
**Email: info@bse.com**

CIRCLE NO. 1130 ON READER SERVICE CARD

## SOFTWARE SORTING

# Opt-Tech Sort

High Performance Sort/Merge/Select Utility. Run from the command line or Call as a subroutine to your programs. Supports most languages and filetypes. Unlimited file size, multiple keys and much more! ***New Version 5.0 includes More Speed and More Features!***

MS-DOS or Windows $149 – OS/2 or Unix $249

**(702) 588-3737**
**Opt-Tech Data Processing**
**P.O. Box 678 • Zephyr Cove NV 89448**

CIRCLE NO. 1135 ON READER SERVICE CARD

*AeSORT Sort Merge/Select*
**ULTIMATE SORT SPEED**

New Criteria for the words 'FAST sort'
Up to 12 TIMES faster than others, 10k size.
Versatile, Easy. DOS exe, batch or call subrtn from popular languages. Mult keys & lengths, Ascn/Dscn, output recs, keys or inx's. Fixed, xbase, etc. Also powerful Merge/Select. $159.

**AeSORT Inc.**
**4070 E. Pueblo, Phoenix, AZ 85040**
**(602) 437-2867**

CIRCLE NO. 1140 ON READER SERVICE CARD

## PROLOG TOOLS

# Discover the *Magic* of Prolog...starting at $49

DOS, 16 and 32-bit interpreter, $49.
Interactive tutorial, $75. Edinburgh-standard development system, $149
Expert systems tutorial w/src, $82.

**FREE Catalog • MC/Visa**
**Moneyback Guarantee**

**Amzi!** 40 Samuel Prescott • Stow, MA 01775
508/897-7332 fax -2784 amziod@world.std.com

CIRCLE NO. 1131 ON READER SERVICE CARD

# NO ROYALTIES

**OmniSort** Sorts, Merges, Selects, and Formats DOS, Btrieve and dBase files *FAST*. Runs as a stand-alone program or links with your Assembler, Basic, C, Cobol, Fortran & Pascal programs. Supports unlimited keys and four gigabyte files.

**Regular $129**
**Special $99 until 4/30/94**

**803 786-7367**
**Omniware**
501 Kinsler Road, Blythewood, SC 29016

CIRCLE NO. 1136 ON READER SERVICE CARD

**DiskCloner** Software disk duplication utility *for Windows™*

- DiskCloner is a super-fast software disk duplicator for Windows.
- Reads source disk images into memory - NO MORE DISK SWAPPING
- Saves disk images to hard disk for later use.
- Share disk images over a network for mass duplication.
- Automatically formats destination disks.
- Software Professionals: DiskCloner is ideal for making internal releases on demand.

To order, send a check or money order for $29.95. Include $3 for shipping in the US. Overseas orders add $5 shipping. CA residents add tax.
North Beach Labs, 350 Union no. 402, San Francisco, CA 94133. Tel/Fax 415-693-0570

CIRCLE NO. 1141 ON READER SERVICE CARD

## SECURITY

# FIGHT PIRACY
# PROTECT YOUR SOFTWARE

Since 1986, thousands of companies worldwide have chosen Az-Tech Software as partners in their fight against Software Pirates. Why? Because we offer you proven leaders:

***EVERLOCK software Copy Protection***
***Option Board Safe—Remote Registration***
***New CPU LOCK-CD ROM LOCK and more***
***EVERKEY HARDWARE LOCKS***
***Call for a FREE Demo***
***(800) 227-0644***

Az-Tech Software, Inc.
201 East Franklin, Richmond, MO 64085
(816) 776-2700 Fax (816) 776-8398

CIRCLE NO. 1132 ON READER SERVICE CARD

## SOURCE CODE ESCROW

# SOURCE CODE ESCROW

*It's finally here* — A flexible source code escrow service for independent developers.

- Guarantee future support
- Enhance your professional image
- You retain copyright

*Call today for a complete information package!*

FIDEX AMERICAS CORPORATION

821 Fir Street • Sandpoint, ID 83864
1•800•569•7569 • FAX 208•265•4191

CIRCLE NO. 1137 ON READER SERVICE CARD

## WINDOWS

# Emacs for Windows

***WinEmacs*** is a fully functional Windows 3.1 version of the industry standard program editor Gnu Emacs, version 19.3.

***WinEmacs*** **HAS THESE EXTENDED FEATURES:**
- Separate buffers in different windows
- Menu and drop-down menu bar
- Multiple font size and type support
- Cut and paste mouse support
- Support for text and Binary files
- Support for foreign keyboards
- Clipboard support
- Scroll bars
- DDE and OLE support
- Binds any arbitrary combination of a key and key modifiers to Emacs Lisp code

Contact **Pearl Software** at pearl@netcom.com (e-mail) (510) 273-9795 (voice) or (510) 839-9820 (fax) for more information. Supported version costs $199.

**CALL 1-800-WIN-EMAC**

**— WE ALSO PROVIDE EMACS CONSULTING SERVICES —**

***Pearl Software*** · 320 Lenox Avenue, Oakland, CA 94610

CIRCLE NO. 1142 ON READER SERVICE CARD

# SOFTWARE COPY PROTECTION

We build quality, security and affordability into our **LTLOCK** copy protection software. Evaluate, compare and see what we are offering:

- Installs easily on your IBM-PC Software
- Defeats all disk copy software
- Requires NO special devices or disks
- MSDOS 6 compatible
- Customize options available

| | | |
|---|---|---|
| Evaluation disk | (5 counts) | $25. |
| Starter kit | (100 counts) | $145. |

unlimited counts kit also available

**L-Tech Corporation**
2460 Lemoine Ave
Fort Lee, NJ 07024

Tel: 201-585-8820
Fax: 201-585-9022

CIRCLE NO. 1133 ON READER SERVICE CARD

## TOOLS

# C and C++ DOCUMENTATION TOOLS

- **C-CALL ($69)** Graphic-tree of caller/called function hierarchy, function/files index, function cross-reference.
- **C-CMT ($69)** Creates/inserts comment-blocks (functions/identifiers used) for each function.
- **C-METRIC ($59)** Calculates path complexity, counts lines with comments, code, and 'C' statements.
- **C-LIST ($69)** Lists and action-diagrams, or reformats source into user selected standard formats.
- **C-REF ($59)** Creates local/global/define/parameter cross-reference, C++ class hierarchy tree.
- **Special: C-DOC ($199)** All 5 in 1 DOS (<15,000 lines)
- **C-DOC Professional ($299)** DOS, OS2, Windows Processes 150,000+ lines, enhanced 3-ring binder.

**SOFTWARE BLACKSMITHS INC.**
6064 St Ives Way,
Mississagua, ONT
Canada L5N-4M1

Voice/Fax: (416) 858-4466
Demos/BBS: (416) 858-1916

CIRCLE NO. 1138 ON READER SERVICE CARD

LINGO LANGUAGE™

Why use a language from the 70's for applications of the 90's? Lingo source code is fully portable across different platforms. Contact us 24 hours a day for more information.

**LINGO LANGUAGE • Phone 61(9) 485 7830 • Fax 61(9) 316 2873**

CIRCLE NO. 1143 ON READER SERVICE CARD

## Listing One *(Text begins on page 121.)*

```
(* ----------------------------------------------------------- *(
** search.pas -- Search engine for IDENT program              **
** Trie search algorithm                                       **
** Copyright (c) 1994 by Tom Swan. All rights reserved.        **
)* ----------------------------------------------------------- *)

unit Search;
INTERFACE
uses Common;

{ Return true if Ident is a Turbo Pascal reserved word }
function IsReserved(Ident: IdentStr): Boolean;
IMPLEMENTATION
type
  ResWord = String[14];
  PResWordRec = ^ResWordRec;
  ResWordRec = record
    Word: ResWord;       { Reserved word string }
    Next: PResWordRec;   { List link field }
  end;

var
  Index: array['a' .. 'z'] of PResWordRec;
{ Add word W to list at P }
procedure AddList(var P: PResWordRec; var W: ResWord);
begin
  if (P <> nil) then
    AddList(P^.Next, W)
  else begin
    P := new(PResWordRec);
    if (P = nil) then
    begin
      Writeln('Out of memory');
      Halt;
    end;
    P^.Word := W;
    P^.Next := nil
  end
end;

{ Add word W to global Index }
procedure AddWord(W: ResWord);
begin
  if Length(W) = 0 then exit;
  AddList(Index[W[1]], W)
end;

{ Initialize search engine variables }
procedure Initialize;
var
  C: Char;  { Index[] array index }
begin
  for C := 'a' to 'z' do
    Index[C] := nil;
  AddWord('and');
  AddWord('array');
  AddWord('asm');
  AddWord('begin');
  AddWord('case');
  AddWord('const');
  AddWord('constructor');
  AddWord('destructor');
  AddWord('div');
  AddWord('do');
  AddWord('downto');
  AddWord('else');
  AddWord('end');
  AddWord('export');
  AddWord('exports');
  AddWord('far');
  AddWord('file');
  AddWord('for');
  AddWord('function');
  AddWord('goto');
  AddWord('if');
  AddWord('implementation');
  AddWord('in');
  AddWord('inherited');
  AddWord('inline');
  AddWord('interface');
  AddWord('label');
  AddWord('library');
  AddWord('mod');
  AddWord('near');
  AddWord('nil');
  AddWord('not');
  AddWord('object');
  AddWord('of');
  AddWord('or');
  AddWord('packed');
  AddWord('private');
  AddWord('procedure');
  AddWord('program');
  AddWord('public');
  AddWord('record');
  AddWord('repeat');
  AddWord('set');
  AddWord('shl');
  AddWord('shr');
  AddWord('string');
  AddWord('then');
  AddWord('to');
  AddWord('type');
  AddWord('unit');
  AddWord('until');
  AddWord('uses');
  AddWord('var');
  AddWord('virtual');
  AddWord('while');
  AddWord('with');
  AddWord('xor');
end;

{ Trie search algorithm }
function IsReserved(Ident: IdentStr): Boolean;
var
  P: PResWordRec;
begin
  IsReserved := false;
  if Length(Ident) = 0 then exit;
  DownCase(Ident);
  P := Index[Ident[1]];
  while(P <> nil) do
  begin
    if P^.Word = Ident then
    begin
      IsReserved := true;
      exit
    end;
    P := P^.Next
  end
end;

begin
  Initialize;
end.
```

## Listing Two

```
(* ----------------------------------------------------------- *(
** common.pas -- Various constants, types, and subroutines     **
** Copyright (c) 1994 by Tom Swan. All rights reserved.        **
)* ----------------------------------------------------------- *)
unit Common;
INTERFACE
const
  identStrLen = 64;
  digitSet = ['0' .. '9'];
  upperSet = ['A' .. 'Z'];
  lowerSet = ['a' .. 'z'];
  alphaSet = upperSet + lowerSet;
  identSet = alphaSet + digitSet + ['_'];
type
  IdentStr = String[identStrLen];
{ Return lowercase equivalent of Ch }
function DnCase(Ch: Char): Char;
{ Convert all letters in identifier to lowercase }
procedure DownCase(var Ident: IdentStr);
IMPLEMENTATION
{ Return lowercase equivalent of Ch }
function DnCase(Ch: Char): Char;
begin
  if Ch in upperSet
    then Ch := Chr(Ord(Ch) + 32);
  DnCase := Ch
end;

{ Convert all letters in identifier to lowercase }
procedure DownCase(var Ident: IdentStr);
var
  I: Integer;

begin
  if Length(Ident) > 0 then
    for I := 1 to Length(Ident) do
      Ident[I] := DnCase(Ident[I])
end;

begin
end.
```

## Listing Three

```
(* -----------------------------------------------------------*(
** ident.pas -- Convert key word identifiers in .PAS files.    **
** Converts key words in Pascal listings to lowercase, and     **
** marks them for bold facing. Words are marked using the      **
** symbols <* and *>. For example, <*begin*> is interpreted as **
** a bold faced "begin" key word. A word-processor macro could **
** search for all <* and *> symbols in the resulting file and  **
** replace these with bold face on and off commands.           **
** Copyright (c) 1994 by Tom Swan. All rights reserved.        **
)* -----------------------------------------------------------*)

{$X+}  { Enable "extended" syntax }
program Ident;
uses Dos, Common, Search;
const
  bakExt  = '.BAK';   { Backup file extension }
  tempExt = '.$$$';   { Temporary file extension }
type
  PString = ^String;
  PListRec = ^TListRec;
  TListRec = record
    Path: PString;
    Next: PListRec
  end;
  TState = (
    Reading, Chkcomment, Comment1, Comment2, Stopcomment,
    Stringing, Converting
  );
var
  FileSpec: ComStr;          { Files entered on command line }
  Root: PListRec;            { File name list root pointer }
  DelimitWords: Boolean;     { True to add <* and *> to reserved words }
  CapIdentifiers: Boolean;   { True to capitalize non-keywords }
```

*(continued on page 144)*

**Listing Three** *(Listing continued, text begins on page 121.)*

```
{ Return copy of a string }
function NewStr(S: String): PString;
var
  P: PString;
begin
  GetMem(P, Length(S) + 1);
  if (P <> nil) then
    PString(P)^ := S;
  NewStr := P
end;
{ Return true if InF is successfully converted to OutF }
function ConvertIdents(var InF, OutF: Text): Boolean;
var
  Ch, PushedCh: Char;
  State: TState;
  Identifier : IdentStr;
  function GetCh(var C: Char): Char;
  begin
    if PushedCh <> #0 then
    begin
      C := PushedCh;
      PushedCh := #0
    end else
      Read(InF, C);
    if (C = #13) or (C = #10) then
    begin
      if (C = #13) then
        Writeln(OutF);  { Start new line }
      C := #0           { Ignore new line characters }
    end;
    GetCh := C
  end;
  procedure UngetCh(Ch: Char);
  begin
    PushedCh := Ch
  end;
  procedure PutCh(Ch: Char);
  begin
    if Ch <> #0 then
      Write(OutF, Ch)
  end;

begin
  PushedCh := #0;      { No pushed character }
  State := Reading;
  while not eof(InF) do
  begin
    GetCh(Ch);
    case State of
      Reading:
      begin
        case Ch of
          '('  : State := Chkcomment;
          '{'  : State := Comment1;
          '''' : State := Stringing;
        end;
        if Ch in alphaSet then
        begin
          UngetCh(Ch);
          State := Converting
        end else
          PutCh(Ch)
      end;
      Chkcomment:
        if Ch = '*' then
        begin
          PutCh(Ch);
          State := Comment2
        end else begin
          UngetCh(Ch);
          State := Reading
        end;
      Comment1:
      begin
        PutCh(Ch);
        if Ch = '}' then
          State := Reading
      end;

      Comment2:
      begin
        PutCh(Ch);
        if Ch = '*' then
          State := Stopcomment
      end;
      Stopcomment:
      begin
        PutCh(Ch);
        if Ch = ')' then
          State := Reading
        else
          State := Comment2;
      end;

      Stringing:
      begin
        PutCh(Ch);
        if Ch = '''' then
          State := Reading;
      end;

      Converting:
      begin
        Identifier := '';
        while Ch in identSet do
        begin
          Identifier := Identifier + Ch;
          Read(InF, Ch)  { Note: Don't call GetCh here! }
        end;
        if IsReserved(Identifier) then
        begin
          DownCase(Identifier);
          if DelimitWords then
            Identifier := '{*' + Identifier + '*}'
        end else
        if CapIdentifiers and (Length(Identifier) > 0) then
          Identifier[1] := UpCase(Identifier[1]);
        Write(OutF, Identifier);
        UngetCh(Ch);
        State := Reading
      end
    end
  end;
  if PushedCh <> #0 then  { Write possible pushed last char that }
    PutCh(Ch);            {  sets eof() to true. }
  ConvertIdents := true
end;

{ Convert one file specified in Path string }
procedure ConvertOneFile(Path: PathStr);
var
  Result: Integer;
  BakF, InF, OutF: Text;
  TempName, BakName: PathStr;
  Name: NameStr;
  Dir: DirStr;
  Ext: ExtStr;
begin
  Write(Path);
  Assign(InF, Path);
  {$i-} Reset(InF); {$i+}
  if IoResult <> 0 then
    Writeln(' **Error opening file')
  else begin
    FSplit(Path, Dir, Name, Ext);
    TempName := Dir + Name + tempExt;
    BakName := Dir + Name + bakExt;
    Assign(OutF, TempName);
    {$i-} Rewrite(OutF); {$i+}
    if IoResult <> 0 then
      Writeln(' **Error creating output file')
    else begin
      if ConvertIdents(InF, OutF) then
      begin
        Close(InF);
        Close(OutF);
        Assign(BakF, BakName);
        {$i-}
        Erase(BakF);
        Result := IoResult;      { Throw out IoResult }
        Rename(InF, BakName);
        Rename(OutF, Path);
```

```
        {$i+}
        if IoResult <> Ø then
          Writeln(' **Error renaming files')
        else
          Writeln(' done')
      end else
        Writeln(' **Error processing files')
    end
  end
end;

{ Convert files on global list at Root pointer }
procedure ConvertFiles(List: PListRec);
begin
  if List = nil then
    Writeln('No files specified')
  else
    while List <> nil do
    begin
      ConvertOneFile(List^.Path^);
      List := List^.Next
    end
end;

{ Add file path to list }
procedure ListFile(var List: PListRec; Path: PathStr);
var
  P: PListRec;
begin
  New(P);
  P^.Next := List;
  P^.Path := NewStr(Path);
  if P^.Path = nil then
    Dispose(P)
  else
    List := P
end;

{ Create list of file names from FileSpec string }
procedure ListFiles(var List: PListRec);
var
  Sr: SearchRec;          { Directory search record }
  L: Integer;             { Length of Dir string }
  OldDir: DirStr;         { Old directory upon entry to procedure }
  Path: PathStr;          { Expanded file specification with path info }
  Dir: DirStr;            { Directory component of Path }
  Name: NameStr;          { File name component of Path }
  Ext: ExtStr;            { File extension component of Path }
begin
  GetDir(Ø, OldDir);                { Save current path }
  Path := FExpand(FileSpec);        { Add path info to file spec }
  FSplit(Path, Dir, Name, Ext);     { Separate Path components }
  L := Length(Dir);                 { Prepare to change directories }
  if L > Ø then
  begin
    if (Dir[L] = '\') and (L > 1) and (Dir[L - 1] <> ':') then
      Delete(Dir, L, 1); { Ensure that ChDir will work }
    ChDir(Dir)           { Change to location of file(s) }
  end;
  FindFirst(Path, Ø, Sr);           { Start file name search }
  while DosError = Ø do             { Continue while files found }
  begin
    Path := FExpand(Sr.Name);       { Expand to full path name }
    ListFile(List, Path);           { Add path to list }
    FindNext(Sr)                    { Search for the next file }
  end;
  ChDir(OldDir)
end;

{ Display instructions }
procedure Instruct;
begin
  Writeln('Use -b option to surround reserved words with');
  Writeln('<* and *> for bold-facing in a word processor.');
  Writeln('Use -c option to capitalize non-keyword identifers.');
  Writeln;
  Writeln('WARNING: After conversion with -b, the listing will');
  Writeln('not compile. Use -b ONLY on a copy of original files.');
  Writeln;
  Writeln('ex. IDENT single.pas');
  Writeln('    IDENT -b one.pas two.pas');
  Writeln('    IDENT wild??.pas -b *.pas')
end;

{ Main program initializations }
procedure Initialize;
begin
  Writeln;
  Writeln('IDENT -- (C) 1994 by Tom Swan');
  Writeln('Converts Pascal reserved words to lowercase.');
  Writeln;
  Root := nil;                { File name list is empty }
  DelimitWords := false;      { Normally do not add <* and *> to words }
  CapIdentifiers := false     { Normally do not capitalize other idents }
end;
{ Main program block }
var
  I: Integer;
begin
  Initialize;
  if ParamCount = Ø then
    Instruct
  else for I := 1 to ParamCount do
  begin
    FileSpec := ParamStr(I);
    if (FileSpec = '-b') or (FileSpec = '-B') then
      DelimitWords := true
    else if (FileSpec = '-c') or (FileSpec = '-C') then
      CapIdentifiers := true
    else begin
      ListFiles(Root);
      ConvertFiles(Root)
    end
  end
end.
```

## Listing Four

```
Sub MAIN
StartOfDocument
EditFind .Find = "<*", .WholeWord = Ø, .MatchCase = Ø, .Direction = 1, \
While EditFindFound()
 EditClear
 EditFind .Find = "*>", .WholeWord = Ø, .MatchCase = Ø, .Direction = 1, \
 If Not EditFindFound() Then
  Stop
 End If
 EditClear
 WordLeft 1, 1
 Bold 1
 EditFind .Find = "<*", .WholeWord = Ø, .MatchCase = Ø, .Direction = 1, \
Wend
End Sub
```

**End Listings**

**Listing One** *(Text begins on page 125.)*

```
;;; _AddInstanceItem hook from LISTINST.386

;;; from DDK VMM.INC
InstDataStruc struc
InstLinkF        dd        Ø   ; linked list forward ptr
InstLinkB        dd        Ø   ; linked list back ptr
InstLinAddr      dd        ?   ; Linear address of start of block
InstSize         dd        ?   ; Size of block in bytes
InstType         dd        ?   ; INDOS_Field or ALWAYS_Field -- ignored?
InstDataStruc ends

;;; from LISTINST.INC -- my InstData struct includes caller address
KM_InstData struc
AddInst_Caller          dd      ?
InstDataStruc           { }       ; from VMM.INC
KM_InstData ends

;;; from LISTINST.ASM
oldservice      dd Ø       ; return value from Hook_Device_Service
Inst_Struc_Ptr  dd Ø       ; InstLinkF from InstDataStruc
calladr         dd Ø       ; address of _AddInstanceItem caller
Data_Buf_Addr   dd Ø       ; created with _PageAllocate PG_SYS
Data_Buf_Size   dd Ø
Data_Buf_Handle dd Ø
Inst_Data_Count dd Ø       ; number of instance items seen so far

;;; from LISTINST.AM Sys_Critical_Init handler

;Instancing of the first byte in the 1st MB in order to get all calls to
;_AddInstanceItem befor ListInst_Sys_Critical_Init. The _AddInstanceItem
;service chains the InstDataStrucs together to a sorted double linked list
;via InstLinkF and instLinkB.
;If the LinkF field is -1, no other calls were made.
;If LinkF <> -1, then it represents a call to _AddInstanceItem caused by a
;system.ini - entry "LOCALTSRS= tsr_name". The VMM instances the whole TSR,
;the first 16 Byte represents the MCB of the PSP. So we can determine the name
;of the fully instanced TSR.
        ;;; ...
        mov KM_Instance.InstLinAddr,Ø
        mov KM_Instance.InstSize,1
        mov KM_Instance.InstType,ALWAYS_FIELD
        mov esi,offset32 KM_Instance
        VMMcall _AddInstanceItem <esi,Ø>
        cmp KM_Instance.InstLinkF,-1            ;any LOCALTSRS ?
        je nolocal
        mov esi,KM_Instance.InstLinkF           ;yes, get it
loclp:  mov Inst_Struc_Ptr,esi
        mov calladr,'LTSR'
        call addinst                            ;add instance item to our list
        mov esi,[esi.InstDataStruc.InstLinkF]   ;get next InstDataStruc
        cmp esi,-1                              ;no more strucs?
        jne loclp

nolocal:mov eax,_AddInstanceItem
        mov esi,offset32 myhook
        VMMcall Hook_Device_Service
        mov [oldservice],esi
        ;;; ...
BeginProc   Hooked_AddInstanceItem
; The AddInstanceItem Hook stores the callers address
; and the instance data pointer in the Inst_Data_Buf buffer.
myhook:
        push ebp
        mov ebp,esp
        push [ebp+Øch]              ; Flags
        push [ebp+8]                ; Instance Structure Pointer
        push [ebp+8]
        pop Inst_Struc_Ptr
        push [ebp+4]                ; get caller's return address!
        pop calladr
        call [oldservice]           ; call original _AddInstanceItem
        add esp,8
        pop ebp
        cmp eax,Ø                   ; error in _AddInstanceItem ?
        je exit
        call addinst                ; add Instance Item to our list
exit:   ret
;*****************************************************************************
;addinst - adds an instance item to our list.
;INPUT:     calladr - address of caller of _AddInstanceItem
;           Inst_Struc_Ptr - address of InstDataStruc
;OUTPUT:    hookerr = -1 - error growing Data_Buf
;           hookerr = Ø - all O.K.
;*****************************************************************************
addinst:push edi
        push esi
        push ecx
        push eax
hook2:  mov edi,Data_Buf_Addr
        mov ecx,Inst_Data_Count
        imul ecx,sizeof KM_InstData
        add edi,ecx
        push edi
        add edi,sizeof KM_InstData
        mov ecx,Data_Buf_Addr
        add ecx,Data_Buf_Size
        cmp edi,ecx
        pop edi
        jl hook1
        mov ecx,Data_Buf_Size
        add ecx,1ØØØh
        shr ecx,Øch
        mov edx,Data_Buf_Handle
        VMMcall _PageReAllocate <EDX, ECX, PAGEZEROINIT>
        cmp eax,Ø
        je hookerr
        add Data_Buf_Size,1ØØØh
        mov Data_Buf_Handle,eax
        mov Data_Buf_Addr,edx
        jmp hook2
hook1:  mov eax,calladr             ;save caller's address in buffer
        stosd
        mov esi,Inst_Struc_Ptr
        mov ecx,(sizeof InstDataStruc)/4
        rep movsd                   ;save instance data struc
        inc Inst_Data_Count
hookret:pop eax                     ;next offset pair
        pop ecx
        pop esi
        pop edi
        ret
hookerr:mov hook_err,-1
        jmp hookret

EndProc   Hooked_AddInstanceItem
```

**End Listing**

## Listing Four *(Listing continued, text begins on page 92.)*

```
void CGAGraphDriver::Draw(CDC &dc, CWordMatrix &Grid)
{
    CPen *pPen = (CPen *) dc.SelectStockObject(BLACK_PEN) ;
    for (int row = 0 ; row < m_GridHeight; row++)
        for (int col = 0 ; col < m_GridWidth; col++) {
           if (Grid[row][col] != EMPTY_CELL)  {
              int x1 = col * (CELL_WIDTH + CELL_SPACE) + CELL_SPACE ;
              int x2 = x1 + CELL_WIDTH     ;
              int y1 = row * (CELL_HEIGHT + CELL_SPACE)  + CELL_SPACE ;
              int y2 = y1 + CELL_HEIGHT    ;
              dc.Ellipse(x1,y1,x2,y2) ;
              char buffer[4] ;
              sprintf(buffer,"%d",Grid[row][col]) ;
              dc.TextOut(x1+CELL_WIDTH/4,y1+CELL_HEIGHT/4,buffer,
                                                            strlen(buffer)) ;
              }
           }
    //draw arcs
    for (int node1 = 0 ; node1 < m_NumGraphNodes; node1++)
        for (int node2 = 0 ; node2 < m_NumGraphNodes; node2++)
            if (m_pGraph->GetAt(node1,node2)) {
               int row1, col1 ;
               Grid.Find(node1, row1, col1) ;
               int row2, col2 ;
               Grid.Find(node2, row2, col2) ;
               int x1 = col1 * (CELL_WIDTH + CELL_SPACE) + CELL_SPACE ;
               int x2 = col2 * (CELL_WIDTH + CELL_SPACE) + CELL_SPACE ;
              int y1 = row1 * (CELL_HEIGHT + CELL_SPACE)  + CELL_SPACE ;
              int y2 = row2 * (CELL_HEIGHT + CELL_SPACE)  + CELL_SPACE ;
              if (x1 < x2)
              x1 += CELL_WIDTH ;
              else
                 if (x2 < x1)
                    x2 += CELL_WIDTH ;
                 else
                 if (x1 == x2) {
                     if (Abs(row1 - row2) > 1) { //route around!
                         y1 += CELL_WIDTH/2 ;
                         y2 += CELL_WIDTH/2 ;
                         int x3 = x1 - CELL_WIDTH/2 ;
                         dc.MoveTo(x1,y1) ;
                         dc.LineTo(x3,y1) ;
                         dc.LineTo(x3,y2) ;
                         dc.LineTo(x2,y2) ;
                         continue ;
                         }
                     x1 += CELL_WIDTH/2 ;
                     x2 += CELL_WIDTH/2 ;
                     }
                 if (y1 < y2)
                     y1 += CELL_HEIGHT ;
                 else
                 if (y2 < y1)
                     y2 += CELL_HEIGHT ;
                 else
                 if (y1 == y2) {
                     if (Abs(col1 - col2) > 1) { //route around!
                         if (x1 < x2) {
                             x1 -= CELL_WIDTH/2 ;
                             x2 += CELL_WIDTH/2 ;
                             }
                         else {
                             x1 += CELL_WIDTH/2 ;
                             x2 -= CELL_WIDTH/2 ;
                             }
                         int y3 = y1 - CELL_HEIGHT/2 ;
                         dc.MoveTo(x1,y1) ;
                         dc.LineTo(x1,y3) ;
                         dc.LineTo(x2,y3) ;
                         dc.LineTo(x2,y2) ;
                         continue ;
                         }
                     y1 += CELL_HEIGHT/2 ;
                     y2 += CELL_HEIGHT/2 ;
                     }

                 dc.MoveTo(x1,y1) ;
                 dc.LineTo(x2,y2) ;
                 }

    dc.SelectObject(pPen ) ;
}
//Calculate the length of the chromosome needed to encode
//a drawing of the graph in a grid
UINT CGAGraphDriver::CalcChromosomeLength() const
{
       return m_NumGraphNodes*(GetNumBitsToEncode(m_GridHeight) +
                                          GetNumBitsToEncode(m_GridWidth));
}
UINT CGAGraphDriver::CalcRowAlleleLength() const
{
    return (UINT) GetNumBitsToEncode(m_GridWidth) ;
}

UINT CGAGraphDriver::CalcColAlleleLength() const
{
    return (UINT) GetNumBitsToEncode(m_GridHeight) ;
}
//Return TRUE if node1 is connected to node2
BOOL CGAGraphDriver::Connected(WORD node1, WORD node2) const
{
    return  m_pGraph->GetAt(node1,node2) ;
}
//Returns the  number of connection leaving node
int CGAGraphDriver::GetNumConnections(WORD node) const
{
    int count = 0 ;
    for (WORD i=0;i<m_NumGraphNodes;i++)
        if (i != node && m_pGraph->GetAt(node,i))
            count++ ;
    return count ;
}
//Returns the total number of connections in the graph
int CGAGraphDriver::GetConnectivity()
{
    int count = 0 ;
    for (WORD node1=0;node1<m_NumGraphNodes;node1++)
        for (WORD node2=0;node2<m_NumGraphNodes;node2++)
            if (node1 != node2 && m_pGraph->GetAt(node1,node2))
                count ++ ;
    return count ;
}
void CGAGraphDriver::Stop()
{
    m_Stop = TRUE ;
}
```

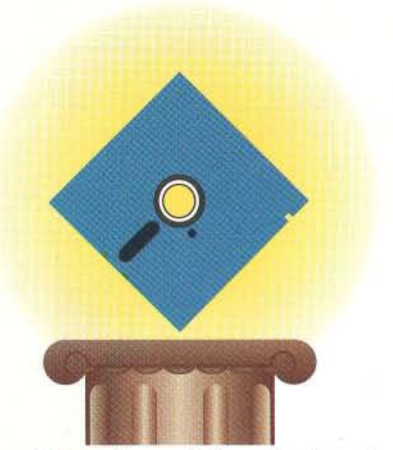
## Listing Five

```
//File: GRAPHGA.H
#ifndef __GRAPHGA_H__
#define __GRAPHGA_H__

//Headers needed for EOS programs
//You need EOS v1.1 to compile this code
#ifndef __BASICGA_H
#include "basicga.h"
#endif

class CGraphDrawerGA : public TBasicGA
{
public:
    CGraphDrawerGA(CGAGraphDriver &driver) ;
    void CreatePopulation(long size, PTIndividual prototype = NULL) ;
    void ExitReport() ;
private:
    BOOL Stop() ;
    void InterGeneration(ulong, PTIndividual, PTIndividual, PTIndividual,
                                                               PTIndividual) ;
    CGAGraphDriver & m_Driver ;
};
#endif
```

## Listing Six

```
//File: GRAPHGA.CPP

#include "stdafx.h"

//Headers needed for EOS programs
//You need EOS v1.1 to compile this code
#include "eos.h"
#include "geno.h"
#include "basicga.h"
#include "nptxgeno.h"
#include "genrepop.h"
#include "gaenviro.h"

//headers specific to graph GA
#include "gdriver.h"
#include "graphga.h"
#include "graphind.h"
#include "grphphen.h"
#include "wmatrix.h"

CGraphDrawerGA::CGraphDrawerGA(CGAGraphDriver &driver)
    : m_Driver(driver)
{
}
//Create the population of individuals
//We use 2 Point Crossover and Elitism
void CGraphDrawerGA::CreatePopulation(long size, PTIndividual prototype)
{
    //Create a genotype with 1 chromosome and 2 point crossover
    //The graph driver is queried to determine the chromosome length
    PTNPtCrossGenotype pGeno =
                   new TNPtCrossGenotype(m_Driver.CalcChromosomeLength(),1,2) ;
    CGraphDrawingPheno * pPheno =
        new CGraphDrawingPheno(m_Driver,m_Driver.GetWidth(),
                                           m_Driver.GetHeight()) ;
    CGraphDrawingInd indiv(pGeno,pPheno);
    m_pPopulation = new TGenReplacePopulation(size,&indiv) ;
    m_pPopulation->SetElitism(2) ;
}
//When the GA is done set the best and worst individuals in the driver
void CGraphDrawerGA::ExitReport()
{
    m_Driver.m_pBest = m_pEnvironment->GlobalFittestIndivid ;
    m_Driver.m_pWorst = m_pEnvironment->GlobalWorstIndivid ;
}
//allow for windows processing!
void CGraphDrawerGA::InterGeneration(ulong, PTIndividual, PTIndividual,
                                          PTIndividual, PTIndividual)
{
    MSG msg ;
    //while there are msgs for status window
    while (PeekMessage(&msg,AfxGetApp()->m_pMainWnd->
                                             m_hWnd,0,0,PM_REMOVE)) {
        TranslateMessage(&msg) ;
        DispatchMessage(&msg) ;
        }
    SetCursor(LoadCursor(NULL, IDC_WAIT));
}
//GA calls this function to determine if it should stop
BOOL CGraphDrawerGA::Stop()
{
    return m_Driver.m_Stop ;
}
```

**End Listings**



To kickstart PowerPC application development, Apple's APDA group has announced a number of Macintosh-based programmer tools for yet-to-come PowerPC-based Apple computers. The "Macintosh on RISC SDK" includes tools for creating new applications or porting existing Macintosh applications for future Apple PowerPC-based PCs. At the same time, Apple introduced the "Macintosh-with-PowerPC Starter Kit" and a comprehensive, self-paced training course entitled *Programmer's Introduction to RISC and PowerPC.* Additionally, Apple is offering Metroworks' native PowerPC development environment, CodeWarrior. Apple PowerPC-based computers are expected to become available in the first half of 1994.

The Macintosh on RISC SDK is an MPW-based cross-development environment that runs on a 680x0 Macintosh, generating native code for Macintosh-with-PowerPC-based systems. When these Macs become available, you can finish the port by testing and debugging your native Mac-with-PowerPC applications. The Macintosh on RISC SDK includes a C/C++ compiler that generates optimized code, PowerPC assembler, two-machine PowerPC debugger, universal system header files for both 680x0 and PowerPC processor-based platforms, MacApp 3.1 (Apple's object-oriented application framework), Apple Installer 4.0 (which is capable of installing either 680x0 or PowerPC environments from a common set of files), MPW Development System 3.3, a PowerPC linker, build tools and scripts, and sample applications for Mac-with-PowerPC.

The Macintosh-with-PowerPC Starter Kit includes detailed technical documentation about both the PowerPC microprocessor and System 7 for Macintosh with PowerPC. Among other information, this kit includes Motorola's *PowerPC 601 RISC Microprocessor User's Manual, Inside Macintosh: PowerPC System Software.*

CodeWarrior is a native development environment for the PowerPC-based and 680x0-based Macintosh that lets you create applications for both platforms using the same source-code base. CodeWarrior comes in three versions: Gold, Silver, and Bronze. Gold, the most comprehensive, includes development releases of C/C++ for the 680x0 Mac and Mac-with-PowerPC, a development release of Pascal for the 680x0 Mac, and C/C++ cross-compilers. Silver supports native PowerPC development only, and will be released when Apple ships Mac-with-PowerPC systems. Bronze supports 680x0 development only.

The Macintosh on RISC SDK, available in prerelease with an automatic upgrade, sells through APDA for $399.00, CodeWarrior Gold (also prerelease) for $399.00, the PowerPC Starter Kit for $39.95, and the Programmer's Introduction for $150.00. Alternately, the tool sets are bundled, selling for $849.00. Reader service no. 20.

APDA
Apple Computer
P.O. Box 319
Buffalo, NY 14207-0319
800-282-2732

A library of encryption tools implemented as linkable object modules and Windows DLLs has been released by AT&T. The library includes RSA, DES, El Gamal public-key, Secure Hash, MD5, and Diffie-Hellman encryption technology.

The code modules are packaged as SecretAgent (DES, El Gamal, and DSA digital signature), SecretAgent II (DES, RSA, and MD5), Surety (DSA), and SecureZmodem (DES using Zmodem protocol).

Prices for code packages containing DSA are $750.00 for DOS/Windows, $1000.00 for Macintosh, and $1250.00 for UNIX. Packages that include RSA sell for $300.00 for DOS/Windows, $400.00 for Macintosh, and $500.00 for UNIX.

The license allows programmers to load the code into two workstations for development purposes. Royalties are required for software distributed to end users. Reader service no. 21.

AT&T Secure Communications Systems
800-203-5563

Visual Xbase, a visual application-development tool for Xbase and C programmers, has been released by Rytech International. The tool includes a 3-D screen designer with an integrated, intelligent data dictionary, workbench, and a flexible, self-optimizing multidialect code generator.

Visual Xbase supports FoxPro 2.5 (DOS and Windows), Clipper 5.2, dBase (IV 2.0), and X2C, an add-on that converts Xbase code to C. In each case, the tool generates code optimized to the selected language. It supports query by example, incremental table searches, filter by example, data-integrity searches, calculated fields, key checking, memory variables, cascaded deletions, and more.

Visual Xbase sells for $495.95. There are no run-time license fees. Reader service no. 22.

Rytech International Inc.
2 Stamford Landing, #100
Stamford, CT 06902
203-357-7812

Mathematica toolkits for electrical engineering, signal processing, control engineering, statistics, finance, and similar disciplines are on the way. The first in the series is the Electrical Engineering Pack, which covers topics ranging from elementary to advanced and includes examples in circuit analysis, transmission lines, and antenna design. The EE Pack also covers Bode, Nyquist and root-locus plots, Smith Charts, and antenna-field patterns. New functions, which extend the original Mathematica product, are specifically aimed at the EE problem domain. Full source code for the examples is also included.

The Electrical Engineering Pack is available for the Macintosh, Windows, and X Window System and is priced at $195.00. Reader service no. 23.

Wolfram Research
100 Trade Center Drive
Champaign, IL 61820
217-398-0747

*Applied Cryptography: Protocols, Algorithms, and Source Code in C,* by Bruce Schneier, has been published by John Wiley & Sons. In its coverage of cryptographic protocols, techniques, and algorithms, *Applied Cryptography* is perhaps the most complete book of its kind. For instance, Schneier (who is a frequent *DDJ* contributor) unravels virtually all block algorithms (including the NSA-backed Skipjack), public-key algorithms (from RSA to cellular automata), one-way hash functions, random-sequence generators, and special algorithms for protocols. In addition, the book provides over 100 pages of published source code for many of these algorithms. Likewise, Schneier's 35-page reference and bibliography section (listing over 900 sources) is of particular value for research.

The 618-page book sells for $44.95 (ISBN 0-471-59756-2). Reader service no. 24.

John Wiley & Sons Inc.
605 Third Ave.
New York, NY 10158
212-850-6000

# C CODE FOR THE PC

*source code, of course*

| | | |
|---|---|---|
| | GraphiC 7.0 (high-resolution, scientific plots in color & hardcopy, contour plots, device independence) | each $370 |
| | X/DOS and Xt/DOS (Xlib with X client and Xt toolkit for DOS; port X code to DOS; Xt/DOS requires X/DOS and 32-bit compiler) | each $300 |
| | ZIP Image Processor & Victor Image Library Version 2.2 (brightness, contrast, merge images, TIFF/GIF/PCX/bin, HP ScanJet support) | $290 |
| | TE Editor Developer's Kit for Windows V3.5 (full screen editor, undo command, word processing; TER for application build-in; no royalties) | $280 |
| | C/C++ Libraries by Code Farms (persistent C structures, ER models, dynamic arrays, database functions, Jolt Award winner; specify C or C++) | $250 |
| | TurboTEX (Release 3.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont) | $250 |
| | Rogue Wave tools.h++ or math.h++ Class Library (extensive docs) | each $240 |
| *NEW!* | Crusher! V2.00 (platform-independent data compression for network transfer; beats PK & LH on binary; directory trees; portable C) | $215 |
| | COMM-DRV (complete interrupt-driven serial communication libraries & device drivers; full source) | $155 |
| | TE Editor Developer's Kit Ver. 3.0 (full screen editor, undo command, multiple windows; with Word Processing $230) | $155 |
| | Minix Operating System (Version 1.5; Unix-like operating system, includes manual; specify 5.25" or 3.5" diskettes) | $150 |
| | XASM (cross assemblers & utility programs; 65xx, 68xx, 80xx; Intel or Motorola hex format; macro preprocessor) | $150 |
| | Delorie GCC for MS-DOS (Version 2.2.2; includes C++, assembler, DOS extender, 387 emulation; complete source code and makefiles) | $150 |
| *Updated!* | Moby Crypto (PGP, DES, Secure Hash, UFC, MDs, Crack 4.1, Lucifer, IDEA, VCR+, large integer packs, tutorials, more; not for export) | $150 |
| | Lisp for DOS (Kyoto Common Lisp and CLISP; KCL includes Lisp-to-C translator for building mixed Lisp/C programs) | $140 |
| | Ibrow (Version 4.1; programmer's Windows-based editor; large files, help, undo/redo, drag-n-drop, function & type tags) | $135 |
| *Updated!* | SCM (portable Scheme in C, IEEE standard, includes JACAL symbolic math package; SCM-4D0/SLIB-1D5/JACAL-1A3) | $100 |
| | PC/IP (CMU/MIT TCP/IP for PCs; Crynwr drivers, NFS server, Bdale mailer, PCRoute/PCBridge, NDIS/ODI drivers, Beholder, more) | $100 |
| | DA (disassembler for Microsoft's New Executable (NE) binary files including Windows .exe, .drv, .dll, and .flt) | $95 |
| | Script Interpreter (a command script interpreter for DOS-based systems; C-like script language; lots of features) | $90 |
| | CELP 3.2c (Federal Standard 1016 Code Excited Linear Predictive voice sampling and encoding; voice over 4.8kbps; Unix code) | $80 |
| | CPPCOMM (C++ serial communications class library for DOS, Windows, OS/2, and NT; includes X/Y/Zmodem) | $75 |
| | ET Neural Net (back error propagation and Kohonen) | $75 |
| | FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++) | $65 |
| | Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic) | $60 |
| | Coder's Prolog (Version 3.0; inference engine for use with C programs) | $60 |
| *Updated!* | PCCTS Version 1.10 (Purdue Compiler Construction Tool Set; like YACC and LEX together with lots of additional features) | $60 |
| | Container Lite V 1.82 (C++ & FLC wrapper emulators; portable, persistent containers of arbitrary data including pointers) | $50 |
| | BigFloat (arbitrary precision floating point arithmetic and functions; includes BCD conversion) | $50 |
| | EZCalc (ASCII algebraic expression evaluator, unlimited parenthesis nesting, symbols, 32 built-in functions, easily extended) | $50 |
| | Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption) | $50 |
| | CLIPS Version 6.0 (rule-based expert system generator; Windows compatible; manuals on disk) | $50 |
| | SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories) | $50 |
| | OBJASM (convert .obj files to .asm files; output is MASM compatible) | $50 |
| | Editor Pack (20 public domain editors; micromacs 3.12, Stevie, Elvis, Moke, mg2a, DTE, Jove, Origami, CE & GRIEF) | $50 |
| | Exceptions for C (Ada-like exception handling for C programs; exceptions for any block; exceptions can be reraised) | $45 |
| | DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud; not for export) | $40 |
| | Database Pack (9 databases – simple to complex: isam, bplus, AVL, SDB, ID, gdbm, Requiem, Ingres89, Postgres) | $35 |
| | COP (poor man's C++; C macro package which implements C++ in C) | $35 |
| *NEW!* | OCT (Object C Translator; essentially Brad Cox's Objective-C Version 4) | $35 |
| | RXC & EGREP Version 2.0 (Regular Expression Compiler and Pattern Matching; finite state machine from regular expression) | $35 |
| | Bison & BYACC (YACC workalike parser generators; documentation; includes C and C++ grammars) | $35 |
| | Spell Pack (6 spelling programs, a hyphenator, 2 utility packs and a 60K word list: Ispell, Microsp, Sp, Cspella, Spell, Dawg, Soundex) | $30 |
| | REGX Plus (Version 3.0, search and replace string manipulation routines based on compiled regular expressions) | $30 |
| | GNU Awk & Diff for PC (both programs in one package) | $30 |
| | Big Number Pack (7 arbitrary precision arithmetic packages in C, one in Fortran but free Fortran-to-C converter is included) | $30 |
| | Crunch Pack (30 file compression & expansion programs; now includes portable ZIP) | $30 |
| *NEW!* | OORT (C++ ray tracing code from the book by Nicholas Wilt) | $30 |
| | OEmacs (full GNU Emacs for DOS and Windows DOS box; C++ support, etags++, lots of .el files) | $25 |
| *NEW!* | CTask Version 2.22d (robust MS-DOS multitasking kernel; C functions run as light-weight processes; mailboxes, interrupts, pipes, etc.) | $25 |
| | PERL for MS-DOS (Version 4.019; C, sed, awk, and shell all rolled into one language; includes hardcopy docs) | $25 |
| *Updated!* | FLEX Version 2.4.3 (fast lexical analyzer generator; new, improved LEX) | $25 |
| | GNU RCS (FSF's version of the Revision Control System; like Unix's SCCS only better; keeps track of software development) | $20 |

## Data

| | |
|---|---|
| Moby Thesaurus II (6,000 root words, 2.5M synonyms, "common sense", concept related searches) | $500 |
| Moby Pronunciator II (175,000 words & phrases encoded with full IPA pronunciation & emphasis points) | $265 |
| Moby Part-of-Speech II (230,000 words and phrases described by prioritized part(s)-of-speech) | $170 |
| Moby Hyphenator II (185,000 words fully hyphenated/syllabified) | $105 |
| Moby Words II (610,000 words & phrases with Scrabble(tm) word list, place names, baby names, acronyms, core list for spell checkers) | $100 |
| CIA World Bank II Database (13MB of maps, 5.7M vectors; coastlines, rivers, political boundaries; 5.25" HD only) | $35 |
| U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points) | $35 |
| The World Digitized (100,000 longitude/latitude of world country boundaries) | $30 |

## CD-ROMs

| | | |
|---|---|---|
| | BSD/386 (POSIX-compatible O/S; complete development package, full networking, kernel debugger, X11R5, DOS box; complete source code) | $900 |
| *NEW!* | AI CD-ROM (expert systems, neural networks, genetic algorithms, fuzzy logic, linguistics/natural language) | $105 |
| | FontMaster II CD Library (soft fonts for HP and HP compatible laser printers, 36 different type faces; 5,200 bit mapped fonts; 300MB) | $70 |
| *Updated!* | Prime Time for Unix (Volume 3, No. 1, January, 1994; over 6GB of Unix C code) | $60 |
| | Walnut Creek Libris Britannia (over 600MB of the best of British boards; not all source included) | $55 |
| *NEW!* | Mailer's Lookup (9-digit ZIP codes by street address or company name, distances between ZIP codes, phone locations; on-line tool) | $50 |
| | Linux/GNU/X by Yggdrasil Computing (1st production release; run from the CD; TCP/IP & NFS; drivers; MPEG; SCSI support; lots more) | $45 |
| | Walnut Creek C User's Group (Volumes 100 to 364) | $40 |
| | InfoMagic Unix (three public domain Unix systems: 386BSD (version 0.1), Linux (version 0.99.10), and NetBSD) | $40 |
| | InfoMagic Source Code (Berkeley Net/2, MACH, GNU, Interviews, X, Andrew, XFree, Demacs & Winemacs, djgpp, Modula-3, etc.) | $40 |
| | Knowledge Media Multimedia (625MB & 13,000 files; 1,232 sounds, 179 books, 100 movies, 114 stacks, 606 programs, 214 mods) | $35 |
| | Project Gutenberg (literature, historical documents, reference books, census data, religious documents, math constants, etc.) | $35 |
| | Knowledge Media Languages & Operating Systems (640MB of compilers, libraries, and operating systems; source code & executables) | $35 |
| | Walnut Creek XllR5 and GNU (X11R5 with contributed and comp.sources.x, over 120 GNU programs, complete C source) | $35 |
| | Walnut Creek Usenet and Simtel Unix-C (600MB) | $35 |
| | Walnut Creek Giga Games (arcade, simulations, card games, education, trivai, cheat sheets; some source) | $35 |
| | Austin Code Works Internet Warrior #1 (PC Internet tools: Gopher, Wais, Eudora, ph, Nupop, Trumpet, TCP/IP, FAQs, drivers, docs) | $35 |
| *NEW!* | Walnut Creek FreeBSD (Berkeley 32-bit operating system for PCs; bootable) | $35 |
| *NEW!* | Walnut Creek Toolkit for Linux (Slackware distrubtion and complete Linux archive) | $35 |
| *NEW!* | Knowledge Media MegaMedia I and II (images, sounds, movies) | each $30 |
| | Walnut Creek CICA Windows Archive (August 1993) | $25 |
| | Walnut Creek Simtel 20 MSDOS Archive (C source code but lots of other stuff too) | $25 |

*The Austin Code Works*
*11100 Leafwood Lane*
*Austin, Texas 78750-3587 USA*

*much more ... ask for catalog*

*Voice: (512) 258-0785*
*FAX: (512) 258-1342*
*E-mail: info@acw.com*

**Free surface shipping for cash in advance**  **For delivery in Texas add 7%**  **MasterCard/VISA**

*(continued from page 148)*
RenderWare, an interactive 3-D graphics API for Windows released by Criterion Software, supposedly increases Windows 3-D graphics performance, without the need for special 3-D graphics accelerators. Based on 3-D graphics software technology from Canon (Criterion's parent company), RenderWare reportedly enables mid-range workstation performance on a 486/50 PC.

RenderWare provides a device-independent 3-D graphics API, an object-based interface consisting of a small number of object types, and functions such as advanced shading and texturing. Typical applications for RenderWare include multimedia, visual simulation, scientific visualization, CAD, virtual reality, presentation graphics, and entertainment/games.

In addition to Windows, the RenderWare API is available on other platforms such as Macintosh, UNIX (X11), and OS/2. The RenderWare SDK, which is priced from $10,000.00, includes a development library, debugging library, documentation, examples, and demos. Reader service no. 25.
Criterion Software Ltd.
17-20 Frederick Sanger Road
Guildford, Surrey
United Kingdom, GU2 5YD
+44-483-574-325



*Undocumented DOS*, second edition, by Andrew Schulman, Ralf Brown, David Maxey, Raymond Michels, and Jim Kyle has been released by Addison-Wesley. The book, spearheaded by Schulman, who edits *DDJ*'s "Undocumented Corner" column, has been updated to include coverage of MS-DOS 6, Novell DOS, Windows 3.1, the forthcoming "Chicago" operating system (DOS 7 and Windows 4), and more.

Like its predecessor, *Undocumented DOS*, second edition belongs on every PC programmer's bookshelf. The book, with disk, retails for $44.95. (ISBN 0-201-63287). Reader service no. 26.
Addison-Wesley Publishing Co.
1 Jacob Way
Reading, MA 01867
617-944-3700

A set of tools that provide programmers with an API for fax-related applications has been developed by Sofnet. These tools, called the "FaxWorks API," facilitate the development of apps that integrate fax, OCR, scanning, voice, and image viewing. The API is proprietary, however, in that it was developed to support Sofnet fax-related software—FaxWorks Pro LAN, FaxWorks OS/2, FaxWorks ProServer, and so on.

The FaxWorks API forms a protocol layer in which other apps can exchange information. FaxWorks acts as a server when applications ask it to perform tasks such as faxing, scanning, or OCR. It acts as a client, however, when it asks applications for data such as a list of names and fax numbers from a phone book. The FaxWorks API and documentation are available free on CompuServe (GO SOFNET). Reader service no. 27.
Sofnet Inc.
1110 Northchase Parkway, Suite 150
Marietta, GA 30067
404-984-8088

As Ken North pointed out in his article, "Database Development and Visual Basic 3.0" (*DDJ*, March 1993), languages embedded in application programs are becoming more and more common. In the case of Microsoft, Visual Basic for Applications (formerly Object Basic) is a programming tool currently available only for Excel 5.0, but with more application support presumably on the way. WordPerfect has countered with WordPerfect 6.0 for Windows SDK and WordPerfect File Format SDK.

The WordPerfect 6.0 for Windows SDK features WordPerfect's new Writing Tools API, a macro language, and Shared Code 2.0 for Windows—a shared library of routines used by all WordPerfect for Windows software.

The File Format SDK, which is available on a nondisclosure basis, contains documentation defining the WordPerfect 6.0 format and the WordPerfect Graphic File Format.

In related news, Softbridge has announced that SBL 3.0, an implementation of the Basic language for embedding into application software, now supports OLE 2.0 automation. This means that SBL, which has a syntax compatible with Visual Basic, can operate within and across applications.

The WordPerfect 6.0 for Windows SDK and WordPerfect File Format SDK sell for $149.00 each. Reader service no. 28.

Softbridge provides various licensing arrangements for SBL 3.0. Reader service no. 29.

WordPerfect Corp.
1555 N. Technology Way
Orem, UT 84057-2399
800-451-5151

Softbridge Inc.
125 Cambridge Park Drive
Cambridge, MA 02140
617-576-2257

According to Al Stevens in this month's "Examining Room," good help can be hard to find when it comes to creating Windows help systems. Addressing this problem is Masterhelp, a new tool from Performance Software. MasterHelp takes text formatted for printing with Word for Windows and automatically creates a Windows help file, including hypertext jumps. The program also automatically creates Microsoft's Multimedia Viewer files for interactive tutorials and the like.

Among features created by MasterHelp are: a table of contents in a secondary window; a pop-up window which provides an overview of the entire document; a pop-up window that lets you know where you are in the document; and a pop-up window that shows hypertext-related topics. MasterHelp retails for $495.00. Reader service no. 30.

Performance Software Inc.
575 Southlake Blvd.
Richmond, VA 23236
804-794-1012

Manageware for NetWare, a multiplatform tool for managing NetWare-based networks from Hitecsoft, is designed specifically for creating NetWare Loadable Modules (NLMs) and server-based applications. Manageware is based on a specification called the "Network Management Language" (NML) developed by Hitecsoft. Similar to a fourth-generation language, NML provides a more flexible means of accessing network internals than traditional languages. NML is operating-system independent and has built-in network extensions (such as client/server, distributed processing, smart-object architecture, and others).

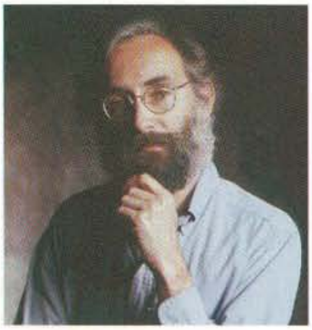
Manageware-based NLMs run under all supported platforms without source-code modification. This means that you can develop and test network-management programs under DOS, then run them as NLMs on the server.

Manageware Version 1.0 is an interpreter (and compiler for the developer's edition) that features a flexible preprocessor, virtual memory management, automatic variable declaration, external function calls, and user-definable functions with local variable declaration and dynamic parameter passing. The tools provide full access to NetWare internals such as binderies, connections, directories, queues, IPX, SPX, and so on.

Manageware for NetWare sells for $895.00. Reader service no. 31.

Hitecsoft Corp.
3370 N. Hayden Road, Suite 123-175
Scottsdale, AZ 85251-6632
602-970-1025

**DDJ**



# *Pentium vs. PowerPC*

It's Pentium vs. PowerPC. That's the simplistic view of what's going on in the area of personal-computer CPUs. DEC and MIPS Technology may see things differently, and this magazine is rarely simplistic about such matters, but this page is where we dumb down to the level of the rest of the computer press. Or even lower. And what's lower than a Lettermanesque Top Ten List? Generous to a fault, we give you two.

**Top Ten Reasons why Pentium will Prevail**

10. Anybody out there using the Dvorak keyboard? You do know that it's been shown to be superior in every way to the ubiquitous Qwerty keyboard, don't you?
9. On the PowerPC you'll have to run Windows and DOS apps under emulation. On Apple's own PowerPC machines, which it is calling Macintoshes, you'll have to run Macintosh apps under emulation. Emulation is slow. Emulation is an unnecessary layer of complexity. Emulation is evil.
8. The subliminal message. It probably wasn't a good marketing decision to call the technology behind the PowerPC "RISC."
7. Intel stock keeps going up.
6. In the short run, the ability to run DOS and Windows apps faster than a 486 machine is what will justify buying a new machine. In the short run, Intel wins.
5. In the long run, bet on the company with the deepest pockets. In the long run, Intel wins.
4. Everybody roots for the underdog. And puts their money on the favorite.
3. Compatibility. Compatibility. Compatibility.
2. Did I mention? Computer buyers value compatibility.
1. The Austin factor. Can we be absolutely sure that those Motorola guys won't withdraw the PowerPC from the market the first time *New York Times* columnist William Safire criticizes it? There's something in the water down there.

**Top Ten Reasons why PowerPC will Prevail**

10. It's got significantly faster floating-point performance than Pentium.
9. It's cheaper. By half.
8. Apple and IBM are solidly behind it.
7. Apple and IBM stocks are recovering.
6. If anyone is looking for a bridge from Intel to RISC, and a lot of people are, it's here. The first generation of PowerPC machines will run existing apps *under emulation* at speeds comparable to existing mid-range to high-end PCs. Native apps will be considerably faster. Early indications are that the emulations will be very solid.
5. Precedent. IBM's RS/6000 workstations haven't done too badly, and PowerPC is the migration of the RS/6000 processor technology to the personal-computer market.
4. The portable edge. The portables market is critical, and by releasing a Pentium chip that won't work in portables, Intel has given PowerPC a huge head start in portables.
3. Price. Price. Price.
2. Well, *do* computer buyers value compatibility? I mean, even if it costs them something? When have they ever had to pay for it? How much are they willing to pay for it?
1. The Clinton clincher. The future belongs to those willing to embrace change.

One reason that did not make the second list: It's Intel's turn to be the Evil Empire. No, IBM had the '80s and Microsoft gets the entire decade of the '90s. No honeymoon for Bill.

Michael Swaine

Michael Swaine
editor-at-large

# C, C++ and BASIC programmers, now you get much more than xBase compatible DBMS power.

Thousands of programmers have already discovered how to get dBASE, FoxPro and Clipper compatibility from their favorite language and hardware platform. For example, one customer has C programs running on PC and Sun workstations sharing data with concurrently running FoxPro for Windows applications. You see, CodeBase technology is simply the best way to add multi-user xBase compatible DBMS power to C, C++, or BASIC.

## You still get high speed & small size

CodeBase users really appreciate our small executable size. Unlike SQL engines which are a Meg or so in size, CodeBase 5.1 EXE's can be as small as 45K! You'll also like the speed–with our Intelligent Queries you get the execution speed of C plus stunning query performance from our smart use of available index information.

## *Introducing* CodeControls

*Now formatted data entry in Windows is as easy as point & click!*


Introducing the new CodeControls, a unique set of data-aware custom controls. Now simply drop them into your Windows applications via your favorite visual interface builder.

## NEW–You get formatted data entry

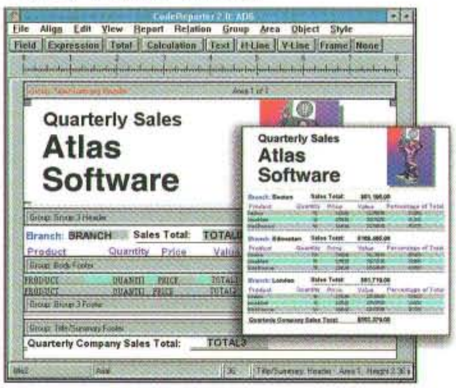
Experienced Windows programmers know **formatted** data entry is difficult to program under Windows. But with our new ***CodeControls***, you can simply 'Point & Click' to design data entry windows for date, numeric, and character information–formatted just the way you want it.

## NEW–Data-aware controls

Our custom controls are ***data-aware,*** so now you can easily build a scrolling list box that's tied to a data file, or look up matching combo box entries–even as the user types.

## *Introducing* CodeReporter 2.0

*Now use the new Instant Report Wizard to create a variety of reports–instantly.*




Introducing the new CodeReporter 2.0, our interactive xBase report writer. We re-designed it with developers in mind, but end-users will love it.

## NEW–You get instant reports

Use our new Instant Report Wizard to quickly create a wide variety of reports–in an instant, then easily refine them as necessary, or let your end users build their own special reports!

## NEW–You get drag & drop

We've added drag & drop capabilities to ***CodeReporter,*** so regardless of whether you want data, totals, or text, simply drop a report object onto your layout screen.

## NEW–You get interactive queries, sorts, & relations

Use CodeReporter to visually design reports easier than ever. For example, you can quickly build query and sort expressions using our calculator-style expression builder. In addition, you can easily relate data files together using our graphical relation builder.

"There's never been a better time to buy into CodeBase technology. You get complete xBase compatibility, full DBMS power, plus an interactive report writer and a set of unique data-aware custom controls for Windows."
**-Ken Sawyer, President, Sequiter Software**

# Buy One, Get Two FREE.

Now when you buy any one of our xBase library products: ***CodeBasic, CodeBase++,*** or ***CodeBase*** (for the language of your choice), you'll get both the new ***CodeReporter 2.0*** AND the new ***CodeControls*** absolutely **FREE**–for a limited time only.

# To Order Now Call 403-437-2410

*Unconditional 90-Day Money-Back Guarantee*

**SEQUITER SOFTWARE INC.**
FAX 403•436•2999
UK Tel. +44-81-317-4321
France +33.20.24.20.14
P.O. Box 575 Newmarket NH 03857-0575